UNIVERSIDADE DA CORUÑA

ESCUELA UNIVERSITARIA POLITÉCNICA

Grado en Ingeniería Electrónica Industrial y Automática

TRABAJO DE FIN DE GRADO

TFG Nº: **770G01A53**

TÍTULO: **ESTRATEGIAS DE CONTROL PARA PLANTA DE LABORATORIO CON RETARDO**

AUTOR: **RUBÉN VALIÑO DÍAZ**

TUTORES: **JOSE LUIS CALVO ROLLE**

JOSE LUIS CASTELEIRO ROCA

FECHA: **SEPTIEMBRE DE 2014**

Fdo.: EL AUTOR

Fdo.: EL TUTOR

ÍNDICE GENERAL

TÍTULO: **ESTRATEGIAS DE CONTROL PARA PLANTA DE LABORATORIO CON RETARDO**

ÍNDICE

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBRERO, S/N

15405 - FERROL

DATA: SEPTIEMBRE DE 2014

AUTOR: EL ALUMNO

Fdo.: RUBÉN VALIÑO DÍAZ

Índice de figuras

Índice de tablas

MEMORIA

TÍTULO: **ESTRATEGIAS DE CONTROL PARA PLANTA DE LABORATORIO CON RETARDO**

MEMORIA

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBRERO, S/N

15405 - FERROL

DATA: SEPTIEMBRE DE 2014

AUTOR: EL ALUMNO

Fdo.: RUBÉN VALIÑO DÍAZ

Capítulo 1

Objeto

El presente proyecto tiene como finalidad la puesta en marcha de una planta de laboratorio de forma que sea controlable y regulable mediante una interfaz gráfica. Para ello se realizará un control manual de la misma al igual que un control automático de seguimiento de consigna mediante la implementación de un control PID(Control Proporcional, integral y derivativo). La adquisición da datos para su control se realizará con hardware ”low cost”, en este caso con una placa Arduino Mega. Para la implementación del control como para la creación de las interfaces gráficas se utilizará el software MATLAB.

Capítulo 2

Alcance

Este proyecto pretende llevar a cabo todas las acciones y requerimientos necesarios para la puesta en funcionamiento de una planta de laboratorio y su posterior control y monitorización. Para la adquisición da datos se usará una placa Arduino MEGA como tarjeta de adquisición de datos (DAQ) y unas placas de amplificación y atenuación. La finalidad de la adquisición es ser aplicada al control por computador mediante MATLAB. El procedimiento a seguir será:

* Comprobación y conocimiento del estado actual de la planta de laboratorio.
* Realizar las modificaciones necesarias para su funcionamiento.
* Prueba inicial de la planta de forma manual
* Configuración de los variadores de frecuencia.
* Configuración de los sensores de nivel.
* Comprobación de las válvulas de paso (válvulas proporcionales y todo/nada).
* Comprobación de los sensores de seguridad.
* Implementación de un programa de control manual de todas las variables de entrada.
* Implementación de reguladores PID para el control automático del nivel con los dos depósitos interconectados y de forma independiente.
* Implementación de una identificación mediante mínimos cuadrados recursivos (RLS).
* Implementación de una interfaz gráfica mediante la aplicación GUI de MATLAB para visualizar y monitorizar las variables que constituyen la planta.

Capítulo 3

Antecedentes

El progreso de la industria en los últimos años ha venido ligado al aumento de la automatización, que se vio en la necesidad de introducir computadores como elementos de control. Este sistema de control obliga a acondicionar y tratar los datos antes de poder ser procesados por el computador, así como procesarlos para que sean comprensibles por un actuador. De este modo son necesarios sensores con acondicionamiento o transductores que conviertan las señales del sensor a eléctricas siendo posteriormente acondicionadas y filtradas para que sean comprensibles por el computador. En este punto es donde se hacen necesario el uso de DAQ que en muchos casos eleva el precio de la automatización. La elección de una DAQ se convierte en una de las piezas fundamentales, por tanto se a de elegir una DAQ adecuada al sistema, al número de variables que forman el sistema y las especificaciones requeridas.

3.1. Tarjeta de adquisición de datos (DAQ)

Una tarjeta DAQ es un dispositivo electrónico encargado de hacer posible la comunicación entre sensores y actuadores con un computador. A la hora de elegir el dispositivo han de tenerse en cuenta una serie de parámetros que definen una tarjeta DAQ:

- Número de canales: dependiendo del número y tipo de variables, es necesaria una u otra tarjeta DAQ. En general se distinguen varios tipos de entradas y salidas en función de 2 criterios: si la entrada es analógica o digital, o si es diferencial o referenciada a masa. Las entradas digitales son aquellas que dan un valor todo o nada (0 o Vcc (Voltaje de alimentación)), mientrás que las analógicas miden una tensión continua (digitalizada internamente). Por su estructura, las entradas digitales son mucho más fáciles de implementar y son más baratas. Dentro de las entradas analógicas se pueden encontrar las diferenciales y

las referenciadas a masa. En las entradas referenciadas a masa, el sensor y la tarjeta DAQ están conectados a la misma fuente de alimentación, siendo común su masa, mientras que las diferenciales son flotantes, es decir, no están referenciadas a la masa de la DAQ. Las entradas diferenciales son más difíciles de implementar y más caras, debido a que éstas pueden estar referenciadas a una tensión de referencia elevada o negativa, de forma que es necesario una mayor-circuitería y aislamiento. Las salidas también pueden ser analógicas o digitales, pudiendo ser estas últimas salidas PWM (Señales modulas en ancho de pulso), que permiten modificar el valor eficaz de la señal al variar el tiempo que está a nivel alto y bajo la señal en un determinado periodo. Las PWM dan una salida a altas frecuencias. Dentro de una DAQ se pueden diferenciar tres partes fundamentales:acondicionamiento, que es un circuito encargado de amplificar, filtrar y aislar las señales de entrada; un convertidor analógico digital y digital analógico; y un bus de comunicaciones con el elemento controlador.

Aparte de su estructura interna, hay que conocer una serie de parámetros que definen una tarjeta DAQ:

- Exactitud: es el valor que se desvía una medida del valor real. Esta está causada por los errores cometidos en los amplificadores, S&H , multiplexores, ADC, offsets, errores de linealidad, retardos, etc.

- Precisión: es la capacidad del sistema de producir una misma salida para una misma entrada. En las tarjetas DAQ se ve afectada por los errores y ruidos del sistema.

- Resolución: es el valor mínimo de entrada que provoca un cambio en el valor que lee el computador.

- Velocidad de muestreo: es la velocidad a la que el sistema adquiere y almacena los valores de entrada. Se pueden muestrear varias señales de forma síncrona, con el correspondiente retardo producido por el multiplexor.

Tras tener en cuenta todos estos parámetros que definirán como serán las señales que reciba el computador estas ya estarán disponibles. El computador recibe los datos ya tratados por la tarjeta de adquisición de datos, que son tratados por software informáticos debidamente programados para realizar una función determinada, obteniéndose unos resultados. Estos resultados son tenidos en cuenta para elaborar órdenes a los actuadores para producir una modificación del estado de la planta, en caso de que fuese necesario. Además, estos resultados pueden ser tratados para la elaboración

de informes, monitorización de la planta, seguimiento del sistema, etc. Se suele emplear como computador un un PC aún siendo posible el uso de microprocesadores o microcontroladores.

Para poder trabajar con el computador, es necesario tener cargado el controlador del bus a utilizar (USB, Wi-fi, etc.), para poder tener una comunicación satisfactoria.

A la hora de implementar un control de un proceso industrial en el que no se conoce el módelo matemático que rige la salida de la misma ante una entrada se suele emplear un regulador o controlador PID. A continuación se introducen las principales características de este tipo de control.

3.2. PID (Proporcional, integral,derivativo)

El controlador de procesos por excelencia en la industria es la implentación de un regulador PID. Aunque en la actualidad ya existen nuevas técnicas para la automatización de procesos mediante modelados matemáticos, el uso de reguladores en el control de plantas y procesos industriales no ha perdido su vigencia. La utilidad de los controles PID radica en su fácil aplicación en sistemas en los que no se conoce su modelo matemático (lo más habitual) y, por lo tanto, no se pueden emplezar métodos de diseño analíticos. Conocer la respueste de una planta ante una entrada, a priori, no es fácil y más cuand se trata de plantas complejas en las que influyen multitud de procesos. Estos motivos, la fácil implementacion de un PID y el buen comportamiento del mismo hacen el PID el método básico a conocer por un profesional de la industria.

El regulador o controlador PID ha sufrido diferente cambios a lo largo de su historia, viéndose reforzado su uso con la aparición de los microcontroladores, lo que produjo que el cálculo digital estuviese disponible de forma muy barata para sistemas pequeños, consiguiendo así un gran impacto en el controlador PID.

El control PID es una implementación simple de la idea de la realimentación (figura 3.2.0.1). La realimentación de un sistema permite reducir los efectos de las perturbaciones (señales que modifican el funcionamiento normal de un equipo o maquina) haciendo así el sistema insensible a las variaciones del proceso. La realimentación de la salida permite que el sistema sea capaz de seguir fielmente la señal de entrada y despreciar el efecto de la perturbaciones sobre el sistema.

Antes de conocer el regulador PID debemos tener en cuenta que existen dos tipos de sistemas según tengan o no realimentacíon de la salida siendo su salida completamente distinta en muchas situaciones. Se habla asi d sistema en lazo abierto o sistema el lazo cerrado.

Sistema en lazo abierto. En ellos la señal de salida no influye sobre la señal de

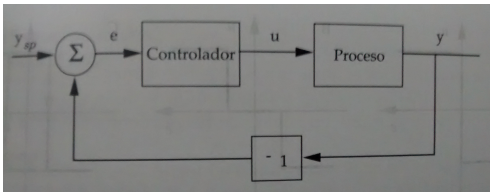


Figura 3.2.0.1 – Diagrama de bloques de un proceso con un controlador por realimentación

entrada. La exactitud de estos sistemas depende de su programación previa. Es preciso se prever las relaciones que deben darse entre los diferentes componentes del sistema, a fin de tratar de conseguir que la salida alcance el valor deseado con la exactitud prevista. Esto significa que no hay retroalimentación hacia el controlador para que éste pueda ajustar la acción de control. La señal de salida no se convierte en señal de entrada para el controlador del proceso. Estos sistemas no son capaces de actuar ante perturbaciones siendo su salida en este caso errónea.

Sistema en lazo cerrado: La toma de decisiones del sistema no depende sólo de la entrada sino también de la salida. Los sistemas de circuito cerrado usan la retroalimentación desde la señal de salida para ajustar la acción de control en consecuencia. Este control usa como acción de control la diferencia entre la entrada y la señal realimentada (error resultante). Esto permite una menor interferencia de las perturbaciones al funcionamiento del sistema. Se consigue así cierta inmunidad ante señales externas. Su implementación es más compleja.

Por lo tanto el regulador PID se trata de un sistema en lazo cerrado dado que existe realimentación de la salida para hayar el error y posteriormente la acción de control.

Como el propio nombre indica el controlador PID está formado por la parte proporcional, integral y derivativa, teniendo cada una de ellas una función específica. A continuación se muestra el diagrama básico de un PID (figura 3.2.0.2).

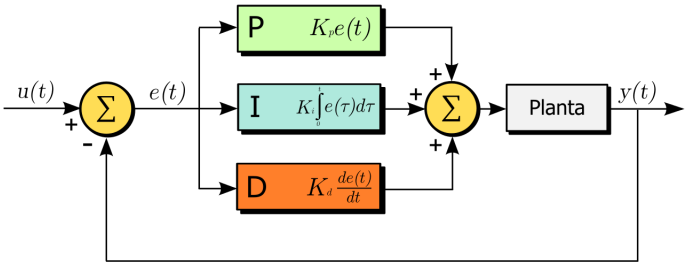


Figura 3.2.0.2 – Diagrama de bloques de un PID

La acción proporcional es el producto de la constante proporcional por la señal de error (diferencia entre la salida deseada y la salida obtenida) de la salida. Se busca conseguir que en régimen estacionario el error del sistema sea o vaya a cero.

$$u = K(Ydeseada - Yobtenida) = kxe$$

Cuando se fije el valor de la constante debe tenerse cuidado de no sobrepasar un valor, dado que existe un límite al partir del cual el sistema sobrepasa el valor prefijado produciendose un fenómeno denominado sobreoscilación.

Existe la posibilidad de implementar un regulador P (regulador proporcional) no siendo la opción de control mas óptima debido a que no se tiene en cuenta el tiempo.

La acción integral del regulador PID tiene como finalidad eliminar o disminuir el error en régimen estacionario provocado por la acción proporcional. La acción integral actúa en el momento que existe una desviación entre la salida y el punto de consigna, integrando esta desviación en el tiempo y sumándola a la acción proporcional. Este es uno de los motivos de por qué son tan empleados los controladores PI.

$$u(t) = Kxe(t) + k_i \int_0^t e(\tau)\, d\tau$$

Regulador PI (Regulador Proporcional e Integral)

La acción derivativa pretende mantener el error al mínimo, corrigiéndolo proporcionalmente con la misma velocidad que se produce, de esta manera evita que el error se vaya incrementando en el tiempo, por tanto se le da al controlador de una capacidad anticipativa ante la aparición del error.

Hay que adaptar la respuesta de control a los cambios en el sistema, ya que una mayor acción derivativa corresponde a un cambio más rápido y por tanto el controlador puede responder acordemente. Por la contra una acción derivativa elevada produce inestabilidad en el sistema. Si su valor es demasiado pequeño la señal de salida oscila demasiado con respecto a la consigna fijada. Se debe buscar una situación de compromiso, un valor óptimo de funcionamiento, aquel que provoque que la variable se aproxime a la consigna con el menor número de oscilaciones posibles.

El regulador PID permite eliminar los errores del sistema en estado estacionario mediante la acción de la parte integral y se anticipa al futuro gracias a la accion derivativa. La mayoría de los procesos se pueden regular de forma eficaz con la aplicación de un regulador PID sin la necesidad de conocer el modelo matemático del sistema.

$$u(t) = K[e(t) + \frac{1}{Ti} \int_0^t e(\tau)\, d\tau + Td\frac{de(t)}{dt}]$$

Regulador PID

La acción de control es así una suma de tres términos que representan el pasado por la acción integral, el presente por la acción proporcional y el futuro por la acción

derivativa. Los parámetros del controlador se llaman: ganancia proporcional K, tiempo integral Ti y tiempo derivativo Td.

Cuando se utiliza un regulador PID uno de los parámetros que hay que tener en cuenta es el periodo de muestreo. Este periódo de muestreo debe ser constante en todo momento y no variar, si se producen variaciones las muestras tomadas del sensor de nivel no estarán separadas entre ellas la misma distancia y la señal obtenida no será la deseada. De este modo el regulador PID no funcionará de forma correcta al no disponer de la señal real. Cada sistema tiene un tiempo de respuesta que debe tenerse en cuenta a la hora de seleccionar el periodo de muestreo, por ejemplo, no debe seleccionarse una periodo de muestro del rango de milisegundos para controlar la temperatura de una habitación.

Capítulo 4

Normas y referencias

4.1. Bibliografía

[1] karl J.Aström, Tore Hägglund, (2009), Control PID avanzado, 1ª ed., NJ: Prentice Hall.

[2] Katsuhiko Ogata , (2010), Ingeniería de control moderna, Pearson

[3] Hebertt Sira-Ramírez, Richard Márquez, Francklin Rivas-Echeverría, Orestes Llanes-Santiago (2005), Control de sistemas no lineales, 1ª ed.,NJ: Prentice Hall.

4.1.1. Soporte digital

[4] Arduino. Edición 1.0.4. Italia: 11 de marzo de 2013, [ref. de 1 de abril de 2013]. “Reference Language”. Disponible en World Wide Web: < *http* : *//arduino.cc/en/Reference/HomePage* >

[5] Arduino. Edición 1.0.4. Italia: 11 de marzo de 2013, [ref. de 1 de abril de 2013]. “Download the Arduino software”. Disponible en World Wide Web: < *http* : *//arduino.cc/en/Main/Software* >

[6] Arduino. Edición 1.0.4. Italia: 11 de marzo de 2013, [ref. de 1 de abril de 2013]. “Examples”. Disponible en World Wide Web:< *http* : *//arduino.cc/en/Tutorial/HomePage* >

[7] Arduino. Edición 1.0.4. Italia: 11 de marzo de 2013, [ref. de 1 de abril de 2013]. “Getting Started w/ Arduino on Windows”. Disponible en World Wide Web: < *http* : *//arduino.cc/en/Guide/Windows* >

[8] Arduino. Edición 1.0.4. Italia: 11 de marzo de 2013, [ref. de 1 de abril de 2013]. “MATLAB Interface for Arduino”. Disponible en World Wide Web: ¡http://playground.arduino.cc/Interfacing/Matlab¿

[9] Inc.. MathWorks. Natick, Massachusetts, Estados Unidos: marzo de 2013, [ref. de 1 de abril de 2013]. “Arduino Support from MATLAB”. Disponible en World Wide Web: $< http : //www.mathworks.es/academia/arduino - software/arduino - matlab.html >$

[10] Inc.. MathWorks. Natick, Massachusetts, Estados Unidos: marzo de 2013, [ref. de 1 de abril de 2013]. “Arduino Software from MATLAB and Simulink”. Disponible en World Wide Web: $< http : //www.mathworks.es/academia/arduino - software/ >$

4.2. Programas de cálculo

En el desarrollo de este proyecto se han utilizado como herramientas los siguientes programas:

4.2.1. Software de programación

* MATLAB R2013A
* Arduino

4.2.2. Editores de texto

* TeXnicCenter

Capítulo 5

Definiciones y abreviaturas

- ⋆ PWM: Señales modulas en ancho de pulso (del inglés, Pulse Width Modulation).
- ⋆ DAQ: Tarjeta de adquisición de datos(del inglés, Data Adquisition).
- ⋆ GUI: Interfaz gráfica de usuario.
- ⋆ LED: Diodo emisor de luz.
- ⋆ V: Voltios.
- ⋆ A: Amperios.
- ⋆ W: Watios.
- ⋆ kW: KiloWatios.
- ⋆ Hz: Hercio.
- ⋆ PC: Ordenador personal (del inglés, Personal Computer).
- ⋆ Perturbación: Señal que tiende a afectar negativamente el valor de la salida de un sistema.
- ⋆ P: Control Proporcional.
- ⋆ PI: Control Proporcional e integral.
- ⋆ PID: Control Proporcional, integral y derivativo.
- ⋆ RLS (del inglés, Recursive-Least-Squares algorithm). Método de identificación mediante mínimos cuadrados.
- ⋆ S&H: Dispositivo de Muestreo y retención (del ingles, Sample AND Hold).

- ⋆ Pin: Patilla para conexiones eléctricas.
- ⋆ N.C: Normalmente cerrado
- ⋆ N.A: Normalmente abierto
- ⋆ init: Secuencia de iniciación.
- ⋆ Nst: Parada variador en rueda libre.
- ⋆ rdy: Variador listo esperando señal de habilitación.
- ⋆ USF: Fallo de subtensión de la red.
- ⋆ Tun: Autoajuste de los parámetros del variador.
- ⋆ Fst: Parada rápida.
- ⋆ Dcb: Frenado por inyección de corriente continua en curso.
- ⋆ Callback: Acción que llevará a cabo un objeto de la GUI cuando el usuario lo active.
- ⋆ Tag: Variable que relaciona un elemento de la GUI con la programación.

Capítulo 6

Requisitos de diseño

Para la puesta en funcionamiento de la planta de laboratorio sobre la que trata este proyecto han de cumplirse unos requerimientos mínimos a la hora del diseño del programa a implementar, como de los elementos a utilizar.

En primer lugar la tensión de alimentación debe ser monofásica, 230 V en alterna. Se necesita una potencia mínima de 1.75 Kw para el correcto y completo funcionamiento de la planta. Esta potencia mínima viene de la potencia de cada variador (0.75 Kw/unidad) más el consumo de la fuente de alimentación de 24 V (250 W). Esta potencia implica la necesidad de una corriente nominal aproximadamente de 8 A.

Para poder llevar a cabo el control es necesario un computador, en este caso un PC con los programas Arduino y MATLAB R2013A instalados.

El equipo utilizado para adquirir las señales de la planta debe ser de bajo coste (low cost).

El programa a diseñar e implementar mediante el software Matlab debe permitir:

- El control de todos los actuadores.
- El control entrelazados de depósitos y bombas.
- El Control manual de la planta de laboratorio.
- El control de la planta mediante un regulador PID.
- El control independiente de los dos depósitos.
- Visualizar el estado de cada variable del sistema.

6.1. Requisitos Físicos del sistema

Al tratarse de una implementación de un programa sobre una planta física también existirán unos requisitos debidos a su constitución y a los elementos que la forman. Estos requisitos dependerán del modo de funcionamiento de la planta. Esto se debe a que en cada modo de funcionamiento el sistema a tratar es diferente.

En el modo de funcionamiento manual se permitirá superar el umbral del 100 % prefijado a la hora de dar potencia a la bomba (no se apagará la bomba aunque se supere el 100 %). A la hora de visualizar nunca se mostrará un valor superior al 100 % debido a las características del sensor de nivel (zona muerta de funcionamiento). Asi mismo el 100 % estará fijado por debajo de los sensores de seguridad, no siendo así utilizado el depósito en su totalidad. Esta configuración permitirá que en caso de querer funcionar con una de las bombas y el depósito contrario en modo manual se pueda alcanzar el nivel del 100 % en dicho depósito. Si la configuración no fuera la mencionada no se conseguiría alcanzar el máximo nivel. Esta limitación se debe a las perdidas producidas en las tuberías y al estar interconectados los depósitos.

En el modo de funcionamiento independiente debe tenerse en cuenta que las válvulas 1 y la todo o nada no se podrán utilizar como perturbación para un único depósito. Se debe a que una apertura o cierre de las mismas influiría del mismo modo a los dos depósitos y por tanto ambos quedarían conectados y perturbados. En el caso de utilizar los dos depósitos a la vez, configurados para estar conectados mediante las válvulas centrales, de producirse el paro del control en uno, la bomba de dicho depósito actuará en dicho momento como perturbación. Este problema se debe a que se produce una succión de agua debido a que se trata de bombas centrífugas.

En el modo de funcionamiento entrelazado es donde se encuentran el mayor número de restricciones. En un primer lugar siempre debe estar una de las dos válvulas centrales (todo o nada o válvula 1) abierta. Si se desea una acción de control más rápida se recomienda tener abiertas totalmente ambas. En caso de disponer de las válvulas en posición cerrada no se realizará el llenado del depósito en ningún momento. En este funcionamiento no se pueden evitar las oscilaciones del sistema, se producirán mayores oscilaciones de la medida sobre todo al inicio, debido al retardo del sistema y al funcionamiento de los depósitos como vasos comunicantes. Se debe llenar el depósito auxiliar a un nivel igual o superior al deseado, retardando al sistema. Otro inconveniente es el rango de funcionamiento, se recomienda un rango del 20 % al 50 % del depósito. Como ya se mencionó anteriormente en esta configuración del sistema es prácticamente imposible realizar un control para conseguir un nivel del 100 % en el depósito . A la hora de querer realizar perturbaciones al sistema debe tenerse en

cuenta que la sección de la tubería de llenado y de perturbación es la misma, por lo tanto el rango en el que se recomienda realizar la perturbación con las válvulas proporcionales restantes no debería superar el 60 %.

Otro punto a tener en cuenta y por el que se obtó por subir el nivel del 0 % es la aparición de de burbujas de aire (Figura 6.1.0.1). En la puesta en funcionamiento del sistema existe aire en las tuberías de las bombas centrífugas, lo que provoca que el funcionamiento de la bomba al inicio no sea el idóneo. Estas perturbaciones influyen en los datos adquiridos mediante el sensor provocando que no sean correctos. Esto se debe a que debido a la aparición de burbujas el nivel del depósito no sube de forma constante y se producen oscilaciones del nivel, lo que provoca que el sensor de nivel no sea capaz de realizar una medida correcta o no sea capaz de realizar la medida. Esta aparicion de aire en los conductos de llenado producirá el fenómeno de la cavitación y a la larga un desgaste irregular de las palas del rodete de la bomba.

Figura 6.1.0.1 – Aparición de burbujas al comienzo del funcionamiento

Otro requisito es la necesidad de que la planta sea educativa y permita conocer como esta realizado su conexionado (Figura 6.1.0.2) y configuración.

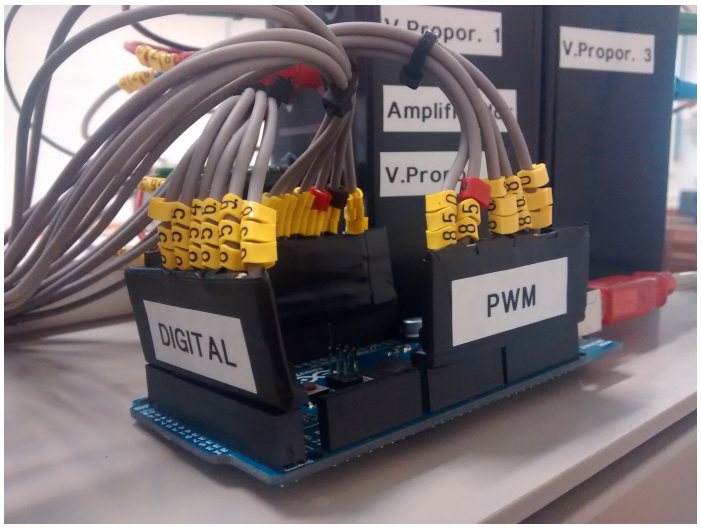


Figura 6.1.0.2 – Conexionado

Capítulo 7

Análisis de las soluciones

7.1. Descripción de la planta

Figura 7.1.0.1 – Planta de laboratorio

La planta (Figura 7.1.0.1) sobre la que se realiza este proyecto permite ser controlada de forma automática mediante el empleo de una DAQ, un acondicionamiento y un pc, teniedo accesible todas variables. Su implementacíon hace prevalecer un sistema

de control mecánico de seguridad al control automático, lo que impide que se produzcan desbordamientos o que alguna de las bombas funcione cuando no existe agua en el tanque. Este sistema mecánico de control está formado por diferentes relés que hacen que se tenga o no la señal de habilitación (si no existe no se activan los variadores) en los variadores de frecuencia y por tanto estos pongan o no en funcionamiento las bombas.

La planta esta constituida físicamente por los siguientes elementos: dos sensores de nivel de ultrasonidos, tres válvulas proporcionales, una válvula pilotada todo o nada, cinco finales de carrera (flotadores), dos bombas centrífugas, dos depósitos y un tanque de almacenamiento de agua. A continuación se describen cada uno de los elementos.

- Depósitos (Figura 7.1.0.2)

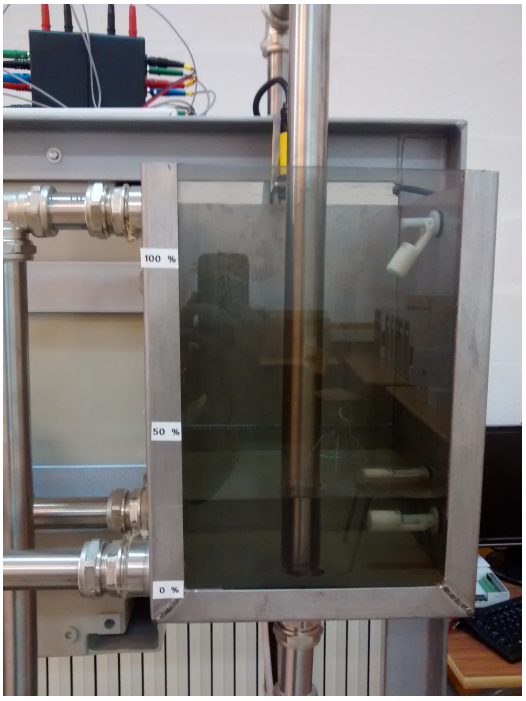


Figura 7.1.0.2 – Rango depósitos

Los depósitos utilizados tendrán la parte delantera de cristal para permitir así visualizar en todo el momento el nivel de agua disponible en cada uno de los depósitos. Estos depósitos tendrán unas tuberías en la parte superior cuya finalidad es funcionar como rebosadero. La bomba hará la aportación por la parte superior del depósito, bajando la tubería hasta la parte inferior del depósito para evitar salpicaduras y por tanto medidas incorrectas del sensor . Como se observa en la imagen el rango del depósito utilizado será de 23cm, dejándose 2cm

en la parte inferior para evitar que al comenzar el llenado, y debido a los remolinos creados, se tomen medidas erroneas. Así mismo en la parte superior se dejarán 5cm para evitar la zona muerta del sensor del nivel y para que actúen los sensores de seguridad.

- Variador de frecuencia ALTIVAR 31 (Figura 7.1.0.3)

Figura 7.1.0.3 – Variador de frecuencia

El variador Altivar 31 es un convertidor de frecuencia para motores asíncronos trifásicos de jaula. Es resistente, de dimensiones reducidas, fácil de instalar y en conformidad con las normas EN 50178, CEI-EN 61800-2, CEI-EN 61800-3, certificaciones UL CSA y la marca CE.

El variador a utilizar estará alimentado a un tensión monofásica de 200-240 V con una potencia máxima de 0.75 KW. Dispone de un panel de control en el cuál existe un visualizador, teclas de navegación en los menús y control local (Marcha/Parada y potenciómetro para ajustar la consigna de velocidad de forma manual).

El variador Altivar 31 dispone de seis entradas lógicas, tres entradas analógicas, una salida lógica/analógica y dos salidas de relé. Las principales funciones integradas son las siguientes:

- Protecciones para motor y variador.
- Rampas de aceleración y deceleración, lineales, en S, en U y personalizadas.

- Más/menos velocidad.
- 16 velocidades preseleccionadas.
- Consignas y posible implementación de un regulador PI.
- Mando 2 hilos/3 hilos.
- Lógica de freno.
- Recuperación automática con búsqueda de velocidad y re-arranque automático.
- Configuración de fallos y de tipos de paradas.
- Memorización de la configuración en el variador.

El variador trabaja en un rango de temperatura desde los -10 a los-50ºC. El equipo viene con protecciones a sobretensiones en la tensión de alimentación, protección térmica contra calentamientos excesivos del variador, protección a cortocircuitos, protección contra sobreintensidades entra las fases de salida al motor y tierra, etc. Es posible controlar el sentido de giro del motor mediante la configuración de las entradas analógicas disponibles.

■ Sensor ultrasonidos (Figura 7.1.0.4)

Figura 7.1.0.4 – Sensor de ultrasonidos

Sensor de la marca banner, modelo Ü-GAGE S18UUA", con salida analógica de 0 a 10 V. Esta constituido mediante un encapsulado para entornos difíciles de trabajo, no siendo el caso. Permite un rango de actuación de -20ºC a 60ºC pudiéndose configurar para tener en cuenta la temperatura de trabajo y realizar

así una compensación de la misma. Dispone de indicadores leds para indicar el estado en el que se encuentra en cada instante. Uno de los leds indica si el sensor está dentro del rango (verde) o si se encuentra fuera del mismo (rojo). El otro indicador led permite saber si la medida tomada es válida (amarillo) o si el sensor esta en modo configuración (rojo).

A la hora de realizar las medidas permite un rango desde los 3 cm hasta 30 cm existiendo una pequeña zona muerta en sus proximidades. Para su correcto funcionamiento debe alimentarse a una tensión de entre 10-30 V en corriente continua. El sensor viene protegido contra cambios de polaridad de la alimentación y tensiones transitorias. El tiempo de respuesta ante un cambio de las variables de medida es de aproximadamente 2.5 milisegundos, existiendo un retardos en el encendido de 300 milisegundos. El error máximo de linealidad como la resolución del equipo es de 1 mm.

- Finales de carrera(flotadores) (Figura 7.1.0.5)

Figura 7.1.0.5 – Final de carrera

Flotador de espuma de propileno con una potencia máxima de 15W. Puede emplearse como contacto normalmente abierto o normalmente cerrado. Sistema de fácil instalación y permite ser utilizado en fluidos con temperatura inferiores a 60ºC. Su función es como final de carrera, abriendo o cerrando el circuito de habilitación de los variadores.

- Bomba centrífuga (Figura 7.1.0.6)

La planta tiene dos bombas centrífugas multicelulares horizontales de la marca Saci y modelo ”MULTINOX-N80-36T). Su potencia máxima de funcionamiento es de 0.6kw (0.8 HP).Esta formada por un motor asíncrono trifásico cerrado de ventilación externa, con un grado de protección IP-44, con una intensidad nominal de 1.2 A. En funcionamiento permite generar un presión de 3.5 bares y aportar

Figura 7.1.0.6 – Bomba centrífuga

un caudal de 4.8 m3/h(80l/min), alcanzando como máximo un altura de columna de agua de 35 metros. El ruido máximo generado por la bomba es de 75 db.

- Válvula Proporcional (Figura 7.1.0.7)

Figura 7.1.0.7 – Válvula proporcional

Las válvulas proporcionales estas formadas por servomotores con posicionadores de la marca Sauter y modelo AXM117S 402. El servomotor tiene una protección IP40, permitiendo dos sentidos de giro regulables mediante la configuración de unos jumpers(permite configurar si a 0 V se quiere la válvula cerrada o abierta). El tiempo que tarda en dar una carrera (pasar de abierta a cerrada) es de 60 segundos siendo la carrera de 4mmm. Dispone de un led que informa si la válvula se encuentra en la posición deseada(luz fija) o se encuentra en movimiento (luz parpadeante). La tensión de alimentación puede ser tanto con-

tinua como en alterna con un consumo de 5VA. Otra configuración posible es la tensión de la señal de control permitiendo tres modos :0-10V, 5.2-10V, 0-4.8 V ejecutándose en los dos últimos casos un porcentaje de la carrera del servomotor. La temperatura máxima a la que puede circular el fluido es de 100 ºC

- Válvula todo/nada (Figura 7.1.0.8)

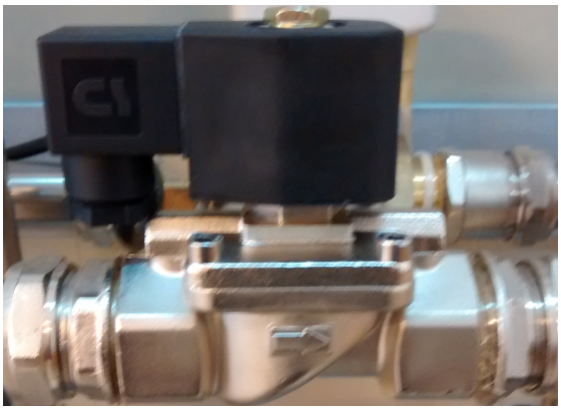


Figura 7.1.0.8 – Válvula todo/nada

La válvula instalada en la planta es del tipo mando directo combinado con 25mm de diametro nominal. Su pilotaje es eléctrico y se encuentra en resposo normalmente cerrada. Para cambiar su estado se debe administrar una tensión de 24 V. La válvula puede realizar como máximo 500 accionamientos/hr. La presión máxima que soporta es de 2 bares y permite el paso de un caudal maximo de 8.82m3/h (147l/min) si el fluido es agua y a una temperatura inferior a 40 ºC. Su consumo eléctrico es de 0.15 A en la conexion y 0.095 A en funcionamiento normal. El módelo utilizado es B 2306C-25 de la marca Electrotaz.

- Aparamenta eléctrica (Figura 7.1.0.9)

  La planta descrita anteriormente cuenta con un cuadro eléctrico y de mando donde se sitúan los variadores de velocidad, fuentes de alimentación, así como toda la aparamenta de seguridad y control necesaria para el correcto funcionamiento de la misma. A cotinuaciòn se hará una pequeña descripción de cada uno de los componentes que la forman.

- Seta emergencia

  Elemento de seguridad que permite la desconexión de la planta en caso de existir un peligro en la misma. Al ser pulsada se produce de inmediato el paro de la planta y queda enclavada la seta, siendo necesario desenclavarla para poder

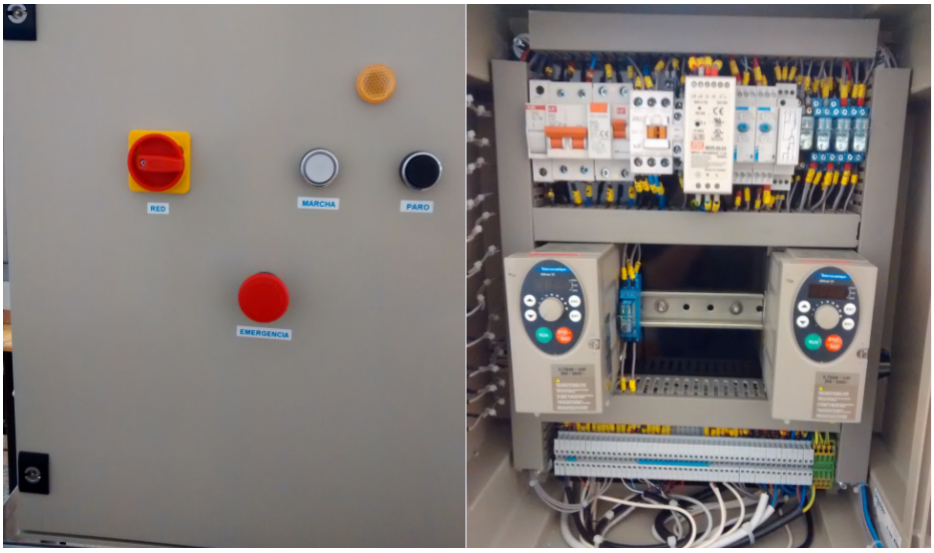


Figura 7.1.0.9 – Cuadro de control

poner de nuevo en funcionamiento la planta. La seta empleada es de la marca Emes con una protección IP20 y tensión de impulso resistida de 2.5 KW.

- Seccionador red

  Elemento de aislamiento y corte que deja al equipo aislado de la red eléctrica a la que se encuentra conectado. Su uso a de realizarse cuando no exista carga, dado que es un equipo de maniobra para funcionar cuando no circula una corriente por el. El seccionador utilizado es de la marca Emes de 2p y 16 A.

- Contactor

  Componente electromecánico que tiene por objetivo establecer o interrumpir el paso de corriente, ya sea en el circuito de potencia o en el circuito de mando, tan pronto como se dé tensión a la bobina.

- Interruptor diferencial

  Dispositivo amperométrico de protección que se desconectan cuando el sistema filtra una corriente significativa a la tierra. Calcula continuamente la suma de vectores de las líneas de corriente monofásicas o trifásicas y, mientras la suma sea igual a cero, permiten que se suministre electricidad el suministro se interrumpe rápidamente si la suma excede un valor predeterminado según la sensibilidad del dispositivo. El diferencial instalado es de la marca LS teniendo una tensión nominal de trabajo de hasta 40 A y una sensibilidad ante las fugas de corriente de 30 mA.

- Fuente de alimentación 24 V

La fuente de alimentación es la encargada de suministrar energía eléctrica a los distintos elementos que componen nuestro sistema electrónico. La planta cuenta con una fuente de alimentación de la marca MEAN WELL, modelo ”MDR-40-24”, conectada a la red eléctrica de 230 V y con una intesidad necesaria de 1.1 A para su funcionamiento, dando a su salida una tensión continua de 24 V y 1.7 A.

- Interruptores de marcha y paro

  Elemento eléctrico que permite o impide el paso de la corriente eléctrica. Los contactos metálicos normalmente separados se unen al ser pulsados y permiten el transcurso de la corriente eléctrica. El interruptor debe ser capaz de realizar el corte con una carga máxima superior a la demandada por el equipo.

- Relés de estado sólido

  Un relé de estado sólido es un dispositivo de conmutación electrónico en el que una pequeña señal de control controla una carga de corriente o tensión más grande. El relé puede estar diseñado para pasar CA o CC a la carga.

- Temporizadores (Figura 7.1.0.10)

Figura 7.1.0.10 – Temporizador

Relé temporizado que permite realizar una conexión o desconexión retardada un periodo de tiempo prefijado. El temporizador empleado es el modelo ”MUR3”de la marca CROUZET. Este temporizador permite ser configurado para trabajar en rangos de tiempo desde 0,1 segundos hasta las 100h de retardo. Esta alimentado a una tensión continua de 24 V. Además este temporizador es multifuncional

permitiendo elegir al usuario el modo de funcionamiento deseado siendo en este caso el modo de funcionamiento H. Dicha función realiza una temporización a la conexión del relé, con lo que se logra que la sálida del temporizador este retardada de la señal de entrada en el tiempo que se ha configurado con anterioridad.

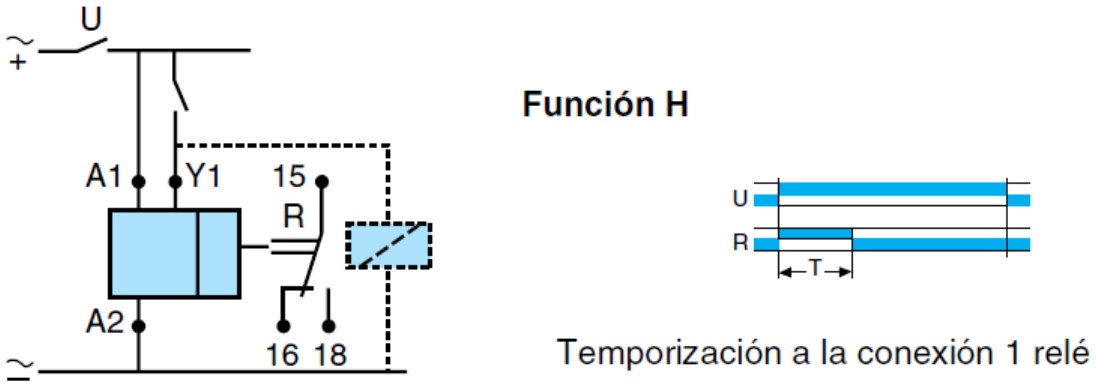


Figura 7.1.0.11 – Esquema temporizador

7.2. Modificaciones de la planta inicial

En primer lugar y antes de entrar en las modificaciones de cableado y configuración se procedio al cambio de la válvula todo o nada dado que la instalada necesitaba una presión de 0.5 bares para ser pilotada. En ningún caso se podría conseguir esta presión en este sistema dado que seria necesario una columna de agua de al menos 5 metros. La válvula instalada finalmente fue la mencionada anteriormente en el apartado anterior.

Inicialmente:

La planta al presionar el pulsador de arranque enciende pero no pasa a estar activa para su uso manual o automático. En este punto los variadores de frecuencia de las bombas se encuentran en el estado ”NST”que significa que el variador se encuentra en parada en rueda libre por entrada lógica. Esto se debe a que debido a su constitución el variador busca seguridad por si existiera un corte en el cable de habilitación por ello al encender debe tener tensión. Para entrar en modo de funcionamiento normal RDY”debe producirse un pulso y posteriormente mantenerse la señal de habilitación a nivel alto (24V).

Solución arranque: Se mantiene la entrada a nivel alto (24V) para no entrar en

modo ”NST”, en este momento el variador pasa al estado RDY.esperando el flanco de habilitación. Este flanco es producido por los temporizadores instalados en la planta. Al temporizador llega la alimentación de 24v y mediante su configuración (función H) se consigue que se produzca un flanco en la señal de habilitación y posteriormente quede a nivel alto de nuevo.

En un primer lugar para tomar contacto con la planta y debido al mal conexionado de los temporizadores se procede a producir el flanco con la seta de emergencia, probando los variadores configurados con señal de entrada(frecuencia) de forma manual. Se comprueba que el sensor final de carrera inferior (FN5) está mal configurado.

Solución final de carrera FN5: Se configura el final de carrera como normalmente abierto (N.A) anteriormente configurado como (N.C). De la misma forma se realiza con los finales de carrera (FN1,FN2).

Solución de temporizadores y de la señal habilitación: Como se comentó anteriormente las señales de habilitación estaban mal configuradas por varios motivos:

1. No permitía que el variador de frecuencia estuviera a la espera de la consigna.
2. La parada del sistema por desbordamiento de uno de los depósitos estaba conectada a los dos temporizadores siendo la desconexión del variador retardada por los temporizadores por el tiempo configurado. Esto permitía que se pudiera producir desbordamiento de los depósitos .
3. Los finales de carrera paraban las dos bombas, cuando cada uno debería deshabilitar el variador de la bomba correspondiente. Así podía producirse el hecho de que un depósito estuviera vacío y se desconectará su bomba.

Para corregir los errores comentados con anterioridad se opta por un nuevo conexionado de la señales de habilitación quedando configurado de la siguiente manera:

Los temporizadores serán independientes, uno para cada bomba y variador permitiendo el uso de la planta como dos depósitos independientes. Se realiza el conexionado para que los variadores entren en modo de funcionamiento normal mostrando en su display ”0.0”. Cada uno de los finales de carrera superior (FN1 y FN2) deshabilitarán los variadores de las bombas 1 y 2 respectivamente. Este cambio implica la utilización de un nuevo relé de estado sólido denominado (K6) para hacer independiente la habilitación de los variadores.

7.3. Elementos externos

Para poder realizar el control sobre la planta de laboratorio es necesario adaptar las señales de salida o entrada a la misma a los rangos necesarios de trabajo, así como

disponer de un equipo que permita adquirir dichas señales para ser posteriormente procesadas mediante el computador (PC).

Se deben generar señales de 0 a 10 V para controlar o actuar sobre los equipos y al mismo tiempo se deben leer señales del mismo rango procedentes de los sensores de nivel. El equipo a utilizar solo permite señales en el rango de 0 a 5 V, por tanto, se emplearan amplificadores y atenuadores de señal. Por otro lado el equipo que se utilizará para la adquisición de dichas señales será una placa Arduino MEGA, un hardware de bajo coste que permite interactuar de forma precisa con la planta de laboratorio.

Se podría obtar por otro hardware de bajo coste como es el caso de la placa Raspberry Pi. La Raspberry Pi es una computadora completamente funcional, mientras que Arduino es un microcontrolador, el cual es sólo un componente de una computadora. Aún así Arduino es muy versátil a la hora de trabajar con proyectos de electrónica (nuestro caso), ya que contiene diferentes pines de entrada y salida lo que permite conectar una infinidad de sensores y actuadores. Otro punto a tener en cuenta es que su conexión con el software Matlab esta mas desarrollada, permitiendo realizar mayores funciones. Por estos motivos y por el conocimiento previo de la placa Arduino se utilizo esta.

7.3.1. Amplificadores y Atenuadores

Como se mencionó con anterioridad es necesario amplificar y atenuar las señales de la planta. Para ello se dispone de unas placas que mediante unos jumpers permiten seleccionar su ganancia. Estas placas están introducidas dentro de unas cajas (Figura 7.3.0.12) en las cuales se dispone de acceso a todas las señales. En el interior van unos amplificadores que serán alimentados desde la patilla de 5 V de la placa Arduino MEGA. Se ha optado por utilizar las placas de forma que el amplificador 1 tenga ganancia 0.5 por tanto atenuará la señal mientras que el amplificador 2 tenga ganancia 2 y por tanto amplificará la señal de salida. Por tanto la configuración queda de la siguiente forma:

- El amplificador 1 tendrá a su entrada la señal proveniente del panel de conexiones correspondiente al sensor de nivel del depósito de la izquierda o derecha, teniendo dicha señal un valor de 0 a 10 V. Este amplificador proporcionará a la salida una tensión que variará de 0 a 5V e irá conectada a las entradas analógicas del Arduino.

- Al amplificador 2 de cada caja se conectara a la entrada la señal proveniente de la placa Arduino correspondiente a la bomba 1 o 2 siendo su valor de 0 a 5 V. Se

Figura 7.3.0.12 – Módulo de amplificación y atenuación

tendrá así a la salida del mismo un señal de 0 a 10 V que ira para al panel de conexiones respectivo a la bomba 1 o 2 .

- Por último se utilizarán dos cajas con dichos amplificadores funcionando en este caso el amplificador 1 y 2 con ganancia 2 para el control de las válvulas. En este caso la señal proveniente de las salidas PWM de la placa Arduino será la entrada de ambos, teniendo un rango de 0 a 5 V y se obtendrá a la salida del amplificador una tensión de 0 a 10 V que ira a la válvula correspondiente del panel de conexiones.

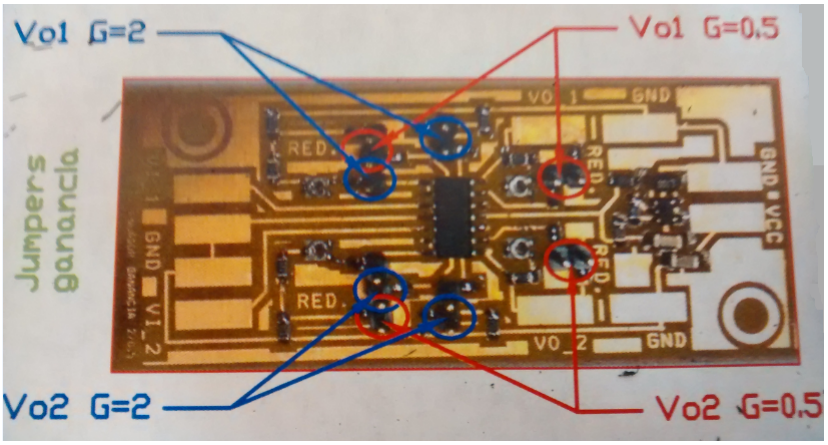


Figura 7.3.1.1 – Placa Amplificación y Atenuación

7.3.2. Arduino MEGA

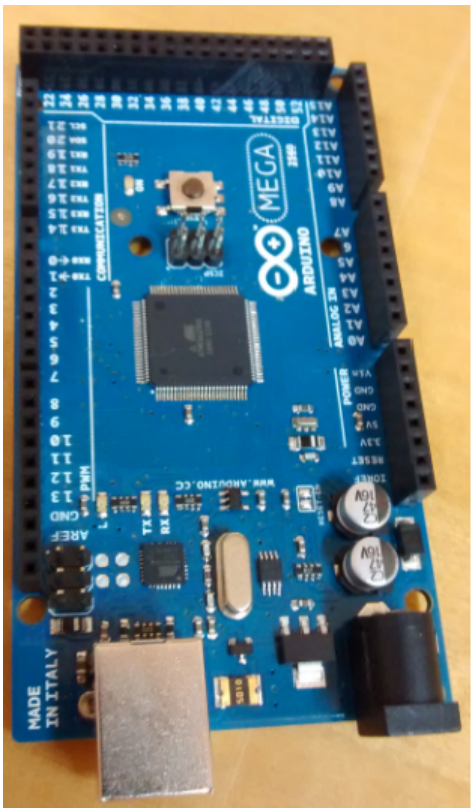


Figura 7.3.2.1 – Placa Arduino

Arduino es una placa hardware libre que incorpora un microcontrolador reprogramable y una seria de pines hembra .En cuanto al software es un entorno de desarrollo gratis, libre y multiplataforma que permite grabar en la memoria de Arduino las instrucciones que se desean ejecutar.

El voltaje de funcionamiento de la placa Arduino es de 5V, pudiendo obtenerse esta

tensión de dos maneras: mediante una conexión a una fuente externa o mediante una conexión a un ordenador con un cable USB, siendo en ente caso alimentado mediante cable USB con una tensión de 5 V y 500 mA de corriente.

A la hora de utilizar Arduino (Figura 7.3.2.1) como una tarjeta de adquisición de datos (DAQ) es importante tener en cuenta el número de entradas y salidas tanto digitales como analógicas, en el caso de la placa Arduino Mega se dispone de :

* Entradas salidas digitales Estas entradas y salidas son pines hembras (GPIO, pines de propósito general) situados en los bordes de la placa, sirviendo cada pin de entrada o salida indistintamente .Los pines funcionan a una tensión de 5(V) pudiendo recibir o enviar un máximo de 40 mA.

  La placa Arduino MEGA cuenta con 54 pines de entradas /salidas digitales (se pueden usar 14 de estas como salidas PWM).

  Estas entradas se utilizarán en este proyecto para adquirir el valor de los finales de carrera (flotadores) y saber si están a un nivel alto o bajo, asi como para gestionar la activación de la válvula todo o nada existente en la planta.

* Entradas analógicas La placa Arduino también dispone de varias entradas analógicas que funcionan en un rango de 0-5 (V).Sin embargo trabaja internamente con un número limitado de valores digitales por lo que existe una pérdida de información. La placa Arduino trabaja con una resolución de 10 bits, es decir que codifica la información de entrada en un numero entre 0-1023. La placa aArduino MEGA cuenta con 16 entradas analógicas.

  Será en estos pines de entrada donde se adquieran los valores de los sensores de nivel posteriormente a la etapa de atenuación necesaria para bajar dichos rangos de 0 a 10 V a un rango válido para Arduino de 0-5 V.

* Señales PWM La placa Arduino no dispone de salidas analógicas propiamente dichas, debido a que el Arduino no es capaz de manejar este tipo de señales .Sin embargo los pines de salida digitales marcados como PWM son capaces de producir una salida promedio entre 0-5 (V).Esta “salida analógica” se consigue gracias a la modulación por ancho de pulso PWM, cuyo funcionamiento se basa en producir pulsos de frecuencia constante (490hz) de duración variable en función de la tensión de salida deseada .La frecuencia de los pulso se puede variar siendo por defecto de 490 Hz (utilizada en el proyecto) hasta una frecuencia de 60 KHz en alguno de los canales. De este modo si la duración del pulso es de medio periodo el valor eficaz seria la mitad(2,5V).Cada pin hembra PWM tiene una resolución de 8 bits ,por lo que se pueden conseguir 256 valores diferentes .

Se utilizarán estas salidas para poder controlar y gestionar la apertura de las válvulas proporcionales existentes así como controlar la velocidad de funcionamiento de las bombas del sistema. El variador de frecuencia tomara un valor de 0 Hz para un valor de 0 en la salida correspondiente un valor de 50 Hz para un valor de 255 de la señal de salida del Arduino pudiendo tomar valores intermedios según el valor de la salida. Por otro lado las válvulas estarán completamente cerradas para un valor de 0 en la salida y totalmente abiertas para una salida de 255 pudiendo tomar valores intermedios.

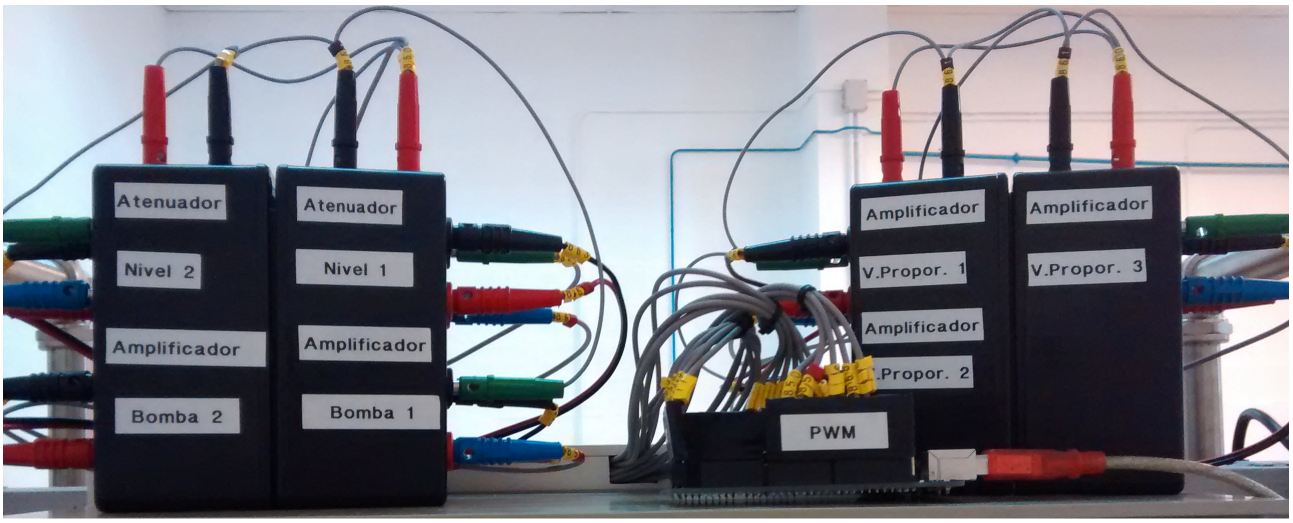


Figura 7.3.2.2 – Conexionado de amplificadores con Arduino

7.4. Arduino como DAQ y configuración con MATLAB

En el caso que nos compete la placa Arduino Mega funciono como DAQ y por tanto seleccionaría una entrada, la muestrearía, mantendría, cuantificaría y codificaría para hacer posible su lectura por el ordenador.

A la hora de realizar la selección de DAQ se opto por la placa ARUDINO dado su bajo coste y sus altas prestaciones en comparación con otros equipos. En algunos sistemas críticos no es conveniente su uso dado que no se conoce como puede responder en situaciones de emergencia además de no existir una certificación ni una garantía de funcionamiento del fabricante. Al no ser la planta de este proyecto un sistema crítico la placa Arduino cumple todas las expectativas y cubre con creces todas las especificaciones requeridas. El número de entradas y salidas digitales, entradas y ”salidas.ªnalógicas son suficiente para monitorizar todas las variables del sistema.

Así mismo el tiempo de muestreo y resolución del equipo son suficientes para el periodo de muestreo y precisión deseada.

Aunque Arduino posee un software libre y gratuito en este proyecto se utilizará como software de programación Matlab dada a sus grandes posibilidades a la hora de adquirir y gestionar datos.

Para poder interactuar desde el software Matlab a la placa Arduino, es necesario seguir una serie de pasos iniciales.

1. Descargar la carpeta “MATLAB support package for Arduino” (“ArduinoIO”), de ($http://www.mathworks.com/academia/Arduino-software/Arduino-Matlab.html$) que consta de una serie de archivos necesarios para llevar a cabo la conexión. Entre ellos hay unos archivos ejecutables por el entorno Matlab y otros por el entorno Arduino.

2. Ejecutar uno de los archivos incluidos en la subcarpeta “PDE” de la carpeta “ArduinoIO” (“adiosrv.pde”,“motorsrv.pde” o “srv.pde”) en el entorno Arduino. Antes de ejecutar cualquier programa, se debe seleccionar en el menú “Herramientas-Tarjeta” el modelo utilizado y en el menú “Herramientas-Puerto Serial” el puerto de comunicación (“COM x”). Estos programas van a estar ejecutándose continuamente en el microcontrolador, permitiendo que la placa Arduino pueda recibir instrucciones de Matlab. Mientras no se cargue otro programa desde el entorno Arduino, no será necesario repetir este paso.

3. Ejecutar el archivo $install_arduino.m$ en Matlab. Con este paso se consigue instalar una toolbox que permite disponer en Matlab de las instrucciones del entorno Arduino, de modo que se programa de forma similar al entorno Arduino pero disponiendo de mayores funciones y herramientas, tales como comandos gráficos, estadísticos, etc.

En este punto ya es posible llevar a cabo la programación en Matlab. En primer lugar, ha de crearse una variable que indique la dirección del puerto al cual está conectado la placa Arduino (a=Arduino(“COMx”)). Al realizar esta instrucción, Matlab realiza una serie de comprobaciones e indica si el puerto está disponible para la comunicación. Si se ejecuta varias veces indicará que el canal de comunicación ya está establecido. Las siguientes instrucciones referentes al Arduino deberán ir referenciadas al puerto de comunicación. La forma de escribir estas instrucciones es $variable_anterior.comando$, por ejemplo, *a.AnalogRead*(5).

Matlab puede comunicarse con Arduino mediante una serie de comandos específicos, como son:

- ⋆ a.pinmode(nº pin, “input/output”): sirve para especificar el modo de funcionamiento del pin.

- ⋆ a.digitalRead(nº pin): sirve para leer la información recibida por el pin especificado.

- ⋆ a.digitalWrite(nº pin, 0 ó 1): da un valor alto o bajo en dicho pin, es decir, 5V o 0V, respectivamente.

- ⋆ a.analogRead(nº pin): sirve para leer la información recibida por el pin especificado. Esta información se leerá en el programa como un número entero comprendido entre 0 y 1023, debido a la resolución de la placa (10 bits=2^{10} resultados =1024 resultados).

- ⋆ a.analogWrite(nº pin,0 a 255):Este comando escribe una serie de pulsos con una determinada frecuencia que da como resultado un valor eficaz comprendido entre 0V y 5V.Este valor eficaz varía en función del valor que tome el segundo término de la función, es decir, que la salida variara 5/255 V por cada unidad en la función .

7.5. Configuración de los variadores de frecuencia

Tras instalar el variador de forma vertical en su posición final se debe configurar diferentes parámetros de funcionamiento. En primer lugar y únicamente en la primera conexión, se debe seleccionar la frecuencia de red a la que trabajará el variador, estando prefijada la frecuencia de 50 Hz. En nuestro caso no es necesario configurar este parámetro dado que se trabajará con 50Hz. En caso contrario se realizaría en el menú SET,en la opción Bfr teniendo el variador parado.

Anteriormente se mencionarón los cambios necesarios en la configuración para poner el variador en el estado Rdy. Los parámetros básicos de funcionamiento que se deben configurar en el variador son: la entrada de habilitación utilizada, rampa de aceleración , rampa de deceleración y la señal control .

El variador dispone de un menú dividido en diferentes apartados según su funcionalidad (Figura 7.5.0.3).

A la hora de realizar la programación hay que tener en cuenta que los códigos de los menús y submenús se diferencian de los códigos de los parámetros por un guion a la derecha. Ejemplos: menú FUn-, parámetro ACC. Tras realizar una modificación de un parámetro del variador este parpadeará.

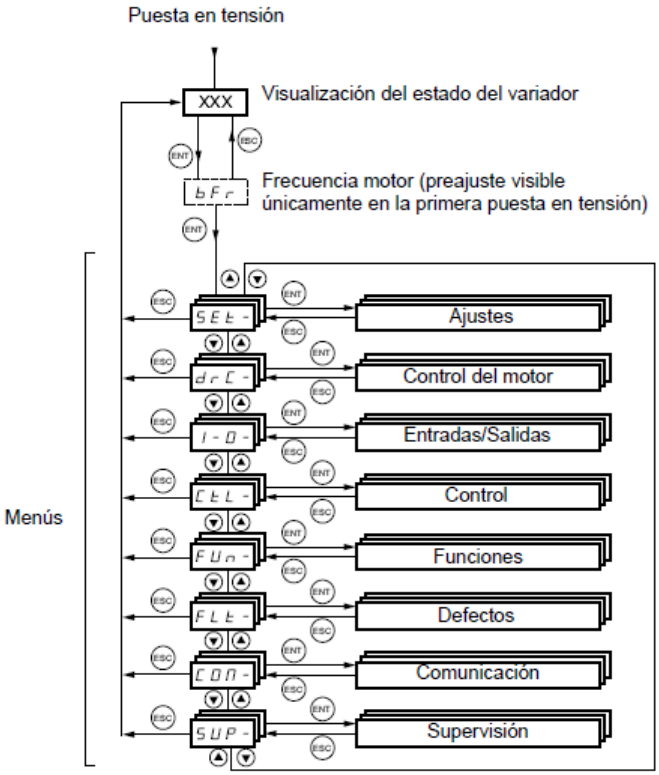


Figura 7.5.0.3 – Menú programación variador de frecuencia

Uno de los primeros parámetros a configurar es la rampa de aceleración, tiempo que tardará el sistema en acelerar desde el valor 0 hasta la frecuencia nominal fijada anteriormente. Esta configuración se realizará en el menú SET y dentro del se accederá a la opción ACC siendo seleccionada una rampa de aceleración de 1.5 s.

A continuación se configurará la rampa de deceleración, tiempo que tarda el variador de pasar de la frecuencia nominal a 0. Se accederá al menú SET y posteriormente a la opción DEC fijando una rampa de deceleración de 1.5 s.

Cuando se realice esta configuración habrá que tener en cuenta la carga de la que se trata, dado que no es lo mismo desconectar un carga inductiva que una altamente inductiva .

Para configurar la consigna del sistema debemos acceder al menú de control Ctl y posteriormente a la función Fr1. Allí se podrán seleccionar diferentes modos. Los variadores utilizados permiten introducir la consigna mediante un potenciómentro instalado en el mismo (opción prefijada y opción AIP). Otra opción y por la que se opta es elegir la opción A11 indicándole así al variador que recibirá la señal de control mediante la entrada AI1. Como comprobación y mediante el menú SUP se puede comprobar cuál es la función de la entrada A11 mediante el submenú A1A. Esta entrada permite indicar el valor de la consigna mediante un rango de tensión de 0 a 10 V dc.

Por último faltaría por seleccionar la señal de habilitación, en este caso se utilizara la entrada lógica LI1 la cuál indica que sentido de marcha se desea, seleccionándose marcha adelante. Cada vez que se inicie el variador se tiene que realizar un autoajuste y para ello se debe producir una transición de nivel bajo a nivel alto (pasar de 0 a Vcc) de la entrada lógica asignada a esta función. Este flanco se produce mediante los temporizadores. Esta configuración se debe realizar en el menú Dcr en la opción tUn.

7.6. Configuración sensores de nivel

Al tener dichos sensores una zona muerta de trabajo y para evitar medidas incorrectas, además de evitar que se interrumpa el funcionamiento de la bomba por la activación del flotador, se optó por bajar la posición del 100 % como se mencionó con anteriorida como requisito físico.

Posteriormente se procedió a la configuración de los mismos de la siguiente manera:

En primer lugar se mantiene pulsado el botón que poseen en la parte superior los sensores para entrar en modo de configuración. Se indica que el sensor en dicho modo mediante una luz roja en uno de los leds. Es en este momento cuando se procede a indicar cuál va a ser el nivel superior por tanto prefijar el 100 % de la medida. Para ello se procede a llenar al depósito y al estar en el punto deseado como el 100 % se pulsa el botón, quedando este valor ya guardado como margen superior. Posteriormente se vacía el depósito hasta el margen del 0 % y se vuelve a mantener pulsado el botón, quedando así configurado el rango de medida del sensor de nivel.

Tras esta configuración el sensor ya estará disponible para realizar las medidas. En los momentos en los que el sensor realiza las medidas del nivel del depósito su aspecto debe ser el de la Figura 7.6.0.4.

Se puede observar que se encuentra uno de los leds en color verde indicando que el sensor se encuentra en funcionamiento y otro de los leds de color amarillo indicando que se está adquiriendo en ese instante el nivel actual del depósito.

7.7. Configuración válvulas proporcionales

Las válvulas proporcionables permiten ser configuradas mediante la colocación de unos jumpers, pudiendo así indicar si se desean abiertas o cerradas cuando su tensión es 0, si se desea un reseteo cada 24 horas o que modo de funcionamiento se quiere. En este caso las válvulas están configuradas para estar normalmente cerradas cuando no tienen tensión (0 V) y se encuentren totalmente abiertas para una tensión de 10 V.

Figura 7.6.0.4 – Lectura del nivel del tanque mediante el sensor ultrasonidos

Dado que la planta no estará encendida de forma continuada (no es su finalidad) el primer jumper que indica si se desea que se produzca un reseteo no tiene mayor importancia, en este caso se optó por no tenerlo activo. Por este motivo el jumper 1 no estará puesto. Los jumpers 2 y 3 estarán puestos, con ello se busca que se realice la carrera completa y esta vaya desde los 0 V hasta los 10 V. Por último como se desea que la bomba este cerrada para 0 V se debe seleccionar la dirección 1 por lo que es necesario quitar el 5º jumper quedando así configurada con al menos los jumpers 2, 3, 5 puestos.

Para poder controlar la válvula en todo el rango y dado que el Arduino Mega solo permite salidas de 0 a 5 V se intercalará el módulo de amplificación descrito anteriormente.

En la Figura 7.7.0.5 se muesta como esta dispuesta la colocación de los jumpers en la válvula proporcional.

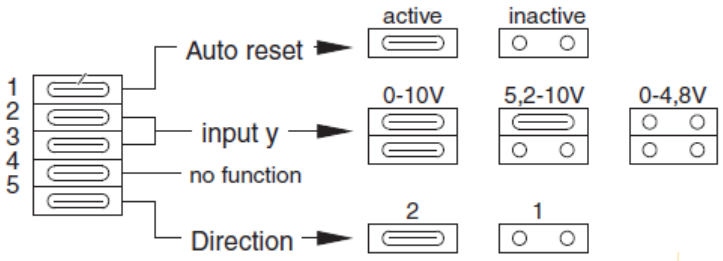


Figura 7.7.0.5 – Configuración de válvula proporcional

7.8. Comprobación sensores de seguridad

Los sensores de seguridad (flotadores) están configurados con sus relés correspondientes de la forma que los flotadores inferiores, que indicarían la existencia de un nivel mínimo de agua para la puesta, estarán configurados como normalmente abiertos. Los sensores superiores estas configurados normalmente cerrados. Con esta configuración se consigue que en el momento que el agua active los flotadores superiores la bomba de ese depósito deja de funcionar inmediatamente, mientras que si no se activa, el funcionamiento de la bomba dependerá del valor de la señal de control.

El único sensor inferior que corta el funcionamiento de las dos bombas es el del tanque principal para evitando asi que las bombas funcionen cuando no existe agua en el mismo. Los sensores inferiores de los depósitos solo indican si existe o no agua en el depósito.

Estos sensores solo influirán la seguridad del sistema para evitar posibles desbordamientos no influyendo en la programación y obtención de la señal de control. Asi mismo serán los encargados de gestionar la señal de habilitación de los variadores en cada momento, cortando dicha señal en los momentos que el agua supere su nivel. Así, aunque exista una señal de control que indique que la bomba debe funcionar, sino se cumple las condiciones de los sensores de seguridad esta no será seguida por el variador .

7.9. PID

Anteriormente se ham comentado los aspectos fundamentales del regulador PID. Como se mencionó, uno de los aspectos fundamentales es el periodo de muestreo, este ajuste se realizará en Matlab mediante el uso de las instrucciones disponibles tic y toc.

Un inconveniente de la parte integral del regulador PID es la saturación de la misma (Figura 7.9.0.6). Cualquier equipo tiene un límite y se puede dar que la señal de control llegue a alcanzar dicho límite, sucediendo en este caso la rotura del lazo de realimentación, eliminandose así la regulación del PID. Al utilizar una parte integral el error seguirá siendo integrado si el diseño del regulador no ha sido el correcto. Esto provoca que el término integral se haga muy grande, a este fenómeno se le conoce con el nombre de Windup.

Para evitar el problema descrito anteriormente se optó por utilizar un algoritmo modificado para el PID, conocido como PID de velocidad o incremental (Figura 7.9.1.1). Este algoritmo calcula primero la velocidad de cambio de la señal de control y enton-

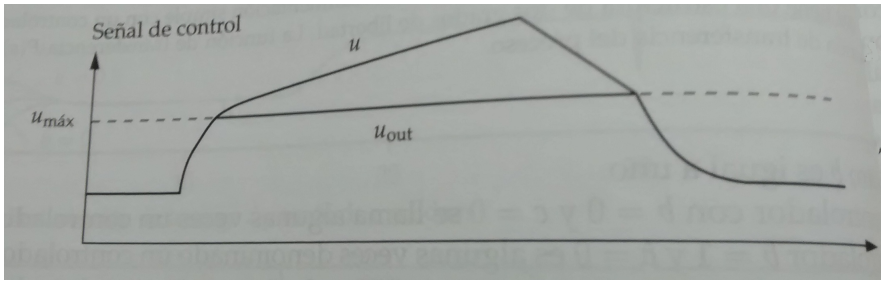


Figura 7.9.0.6 – Saturación del integrador

ces se alimenta a un integrador. El Windup se evita inhibiendo la integración siempre que se produzca la saturación de la salida. Cuando la salida satura, se recalcula el término integral en el controlador de forma que su nuevo valor da una salida en el límite de la saturación.

7.9.1. PID de velocidad o incremental

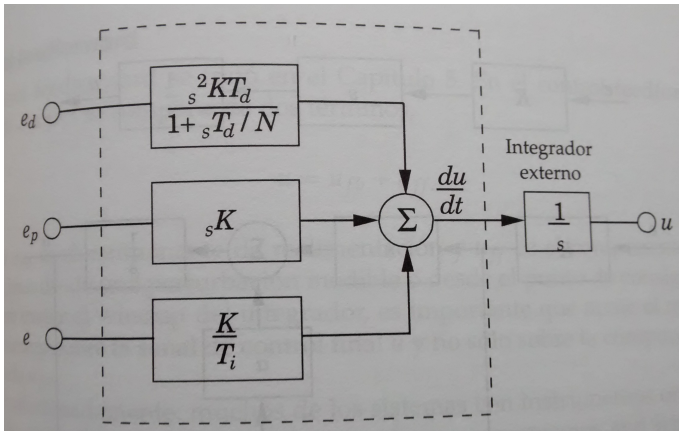


Figura 7.9.1.1 – Diagrama de bloques de un algortimo PID de velocidad

Los algorítmos descritos en la sección antecendetes son llamados “algoritmos de posición” debido a que su salida es la propia variable de control. En ciertos casos, el sistema de control está configurado de forma que la señal de control está manejada directamente por un integrador, por ejemplo un motor. Por tanto, es natural acondicionar el algoritmo de forma que proporcione la velocidad de la variable de control. En este caso, la variable de control se obtiene mediante la integración de su velocidad.

A la hora de implementar este PID en Matlab se realizó mediante una función a la cuál se le pasan los valores de los parámetros del PID (Td, Ti, kp), la consigna y la salida el sistema. Posteriormente y tras los cálculos oportunos el sistema devuelve en valor adaptado para la salida a la bomba mediante Arduino Mega.

```
function [ output_value,Error,Control_signal ] = PID_PROPIO( kp
    ,Ti,Td,input_voltage,Error,Control_signal,nivel_deseado );
%Realiza el controlador PID del sistema
%Para su funcionamiento se le aporta a la función los
    parámetros propios del pid,asi como la salida y la consigna
    del sistema,devolviendo la funcion la señal de control a
    aplicar al sistema
wait_time=0;
T=0.5; %Periódo de muestreo

      start_time=toc; %Almacena el tiempo de inicio de la
          muestra.Instrucciones
      variable_anterior=input_voltage;
      %Con el siguiente bucle se pretende evitar la aparición de
          medidas incorrecta por culpa de las perturbaciones
          introducidas por la bomba
       if input_voltage<(variable_anterior -100)
         input_voltage=variable_anterior;
       end
      nivel_actual=((1023-input_voltage)*100)/1023; %Cálculo del
          nivel
      %Calculo de los errores
      Error(1)=Error(2);
      Error(2)=Error(3);
      Error(3)= nivel_deseado-nivel_actual;
      %Calculo de los pesos de influencia de cada error
      q0=kp*(1+(Td/T));
      q1=kp*(-1 +(T/Ti)-2*(Td/T));
      q2=kp*(Td/T);
      %Determinacion de las señales de control
     Control_signal(1)=Control_signal(2);
     Control_signal(2)=Control_signal(1)+q0*Error(3)+q1*Error(2)+
         q2*Error(1);
      if Control_signal(2)>100
          Control_signal(2)=100;
      end;
      if Control_signal(2)<0
```

```
        Control_signal(2)=0;
    end;
    output_voltage=Control_signal(2)/100; % tensión de salida
    output_value=round((output_voltage*255)) %Escalado de la
        salida para su salida mediante el Arduino Mega

    %Instrucciones para comprobar que se cumplió el tiempo de
        muestreo
    spend_time=toc-start_time; %Comprobamos si el tiempo
        transcurrido permite cumplir con el período de muestreo
    if spend_time>T
        disp('Tiempo de muestreo superado'); %Mostramos la
            causa de fallo

    else
        wait_time=T-spend_time; %Si el tiempo transcurrido es
            menor, se esperará hasta completar T
        pause(wait_time);
    end

end
```

En un primer lugar se realizará un ajuste empírico del regulador PID en cada uno de los modos. Para llevarlo a cabo se fue probando con diferentes valores en los parámetros hasta conseguir una respuesta del sistema satisfactoria. Es un método lento y de prueba-error. Lo bueno de este método es que no implica un alto conocimiento de la prorgramación y estas pruebas pueden ser realizadas por personal no cualificado.

En este caso también se permite que el usuario modifique los parámetros prefijados para que comprueba la variación de la respuesta. Así podra ver in situ los problemas comentados anteriormente y decidir que situación de compromiso toma. Puede elegir entre sistemas con una respuesta rápida y oscilante o por el contrario por un sistema lento y con menor oscilación. Estos parámetros serán diferentes en cada modo dado que al entrar en un modo la configuración de la planta es diferente y por tanto se trata de un sistema diferente en cada caso. Hay que tener en cuenta que la existencia de tuberías, válvulas intermedias o el retardo producido por el depósito auxiliar hacen que el sistema cambie completamente. Por este motivo la salida del sistema es distinta para los mismos parámetros.

7.10. PID mediante Histéresis

Un método de calcular los valores de los parámetros del regulador PID es mediante el funcionamiento del sistema como un relé con una histéresis en el nivel del depósito. Este método nace del método de lazo cerrado propuesto por Ziegler Nichols, conocido como método de oscilación sostenida. A la hora de utilizar este método se pretende conocer los siguientes parámetros:

- Ganancia proporcional crítica (Kc).- Es la ganancia de un controlador solo proporcional, que provoca que el sistema sea oscilatorio (críticamente estable).

- Período de oscilación sostenida (Tc).- Es el periodo de oscilación que se consigue con la ganancia crítica.

El método de oscilación sostenida no es muy empleado ya que puede llevar al sistema a la zona inestable, por ello el uso del método Relé (Figura 7.10.0.2). Este método fue desarrollado por Aström y Hägglud y consiste en llevar al sistema al estado de oscilación con la incorporación de un relé. De este modo se consigue una oscilación sostenida del sistema a partir de la cual se hayan los valores. La consigna que llega a la bomba para su funcionamiento será todo o nada, por tanto la bomba funcionará al 100 % o si no se encontrará apagada. De este modo se calcula la ganancia critica Kcr y el periodo Pcr correspondiente.

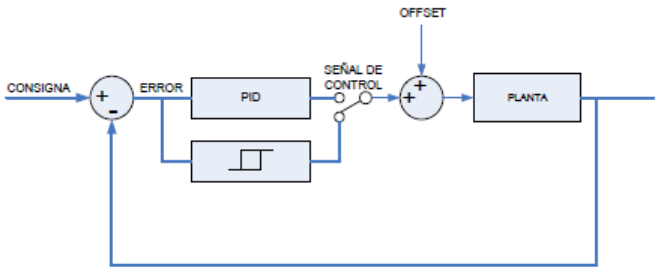


Figura 7.10.0.2 – Esquema método Relé

Se ha de producir con el relé programando una histeresis como la de la Figura 7.10.0.3.

El proceso a seguir será el descrito a continuación:

1. Se configura el programa para realizar una histéresis (offset) alrededor de un valor prefijado.

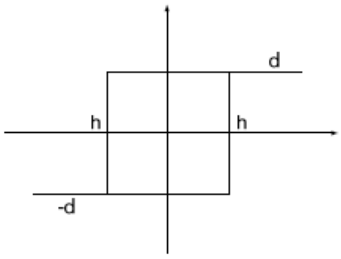


Figura 7.10.0.3 – Histéresis metodo relé

2. La salida del sistema será todo o nada entre dicho rango siendo la salida nula desde cuando se supera el límite superior hasta volver al límite inferior. La salida tomará el valor máximo cuando se encuentre en el recorrido del límite inferior al límite superior.

3. Se deja el sistema en marcha hasta conseguir que la salida del sistema sea periódica (en este caso 6 periodos).

4. Se toman los valores de la amplitud del sistema y el periodo de la señal en el último periodo. Esta señal será similar a la de la Figura 7.10.0.4.

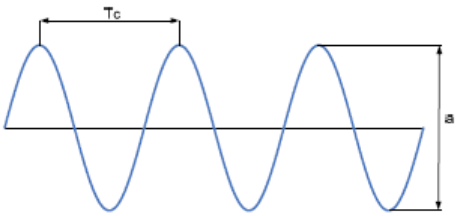


Figura 7.10.0.4 – Salida del sistema en el método relé

5. Se calcula la ganancia crítica del sistema Kcr mediante:

$$Kcr = \frac{4Xd}{\pi x \sqrt[2]{a^2 - h^2}}$$

6. Se calculan los valores de los parámetros mediante las tablas de Ziegler-Nichols para lazo cerrado(Tabla 7.10.0.1).

A la hora de implementar este método mediante Matlab se hace un ajuste. En primer lugar se aplica una factor del 0.1 por el valor hayado del parámetro Td, consiguiendo así que el sistema sea más estable que en el caso de seleccionar los valores hayados directamente. Esto ayuda al correcto funcionamiento de la planta.

Tipo de controlador	Kp	Ti	Td
P	0,5xkcr	∞	0
PI	0,45xKcr	$\frac{1}{1,2}$xPcr	0
PID	0,6xkcr	0,5xPcr	0,125xPcr

Tabla 7.10.0.1 – Tabla parámetros Ziegler-Nichols para lazo cerrado

Cuando se selecciona el método histéresis en el modo de control se le indica al programa que se desea obtener los parámetros mediante dicho método, realizandose así la llamada a la función método relé. Tras ser cálculados se envían al programa general. Posteriormente se llama a la función PID decrita anteriormente a la que se le envían estos parámetros. Su funcionamiento se describe en la Figura 7.10.0.5.

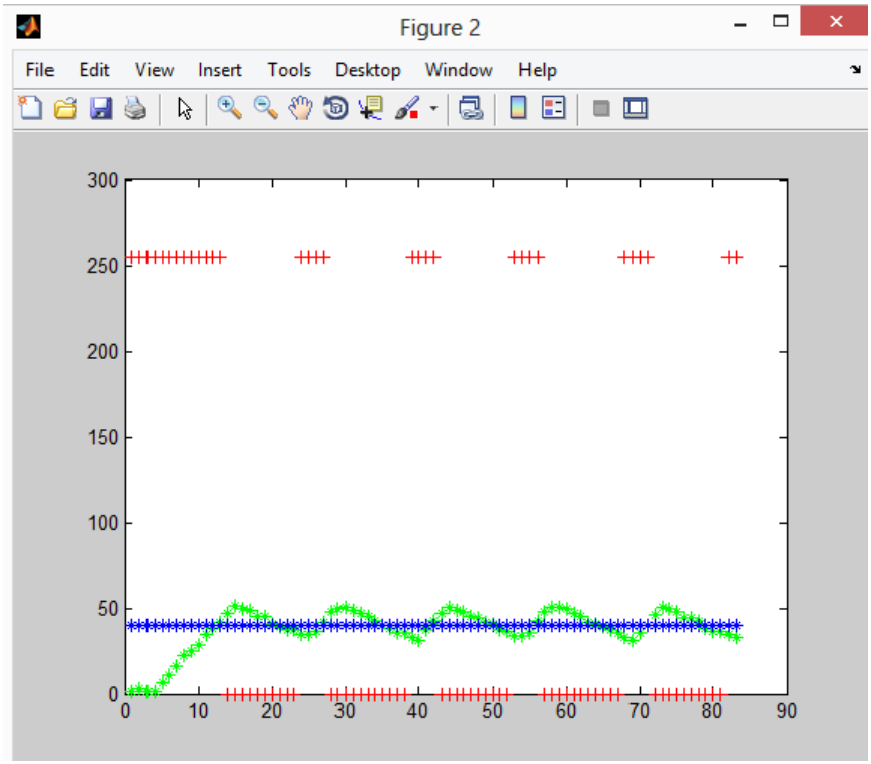


Figura 7.10.0.5 – Funcionamiento método relé en Matlab

A la hora de implementar la función del método relé mediante Matlab se hace un ajuste propio. Se aplica una factor del 0.1 por el valor hayado del parámetro Td, consiguiendo así que el sistema sea más estable que en el caso de seleccionar los valores hayados directamente. Este factor se obtuvo de la realización de pruebas y se comprobo que era el valor apropiado para obtener un sistema relativamente rápido y con poca oscilación. Ya se dijo anteriormente que el valor de Td no debe ser elevado por ello un factor de reducción y no de aumento. Esto ayuda al correcto funcionamiento de la planta.

7.11. Identificación de la planta.

La identificación de sistemas es la aproximación experimental al modelado de sistemas. Consiste en obtener un modelo a partir de observaciones obtenidas directamente del propio sistema que se pretende modelar, en este caso la planta de laboratorio. La identificación de la planta permite obtener una función de transferencia, la cuál indica mediante una aproximación, como es el funcionamiento de la planta para cada una de las entradas y cual será su respuesta. Al tratarse de un sistema discreto, formado por diferentes muestras, la función de transferencia será en el dominio Z.

A la hora de realizar una identificación se debe distinguir en identificación en linea o fuera de línea.

- Identificación en línea (Figura 7.11.0.6)

  En los métodos de identificación en línea la estimación se efectúa usando medidas que se van obteniendo en tiempo real, y normalmente se usan cálculos recursivos.

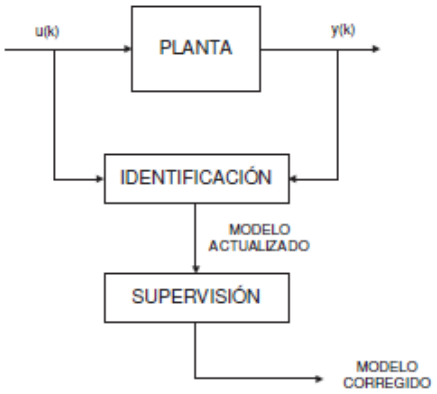


Figura 7.11.0.6 – Esquema de identificación en linea

- Identificación fuera de línea

  En este caso se toman series de medidas y posteriormente, se ajusta el modelo usando para ello todo el conjunto de datos. Este tipo de procedimientos suelen obtener modelos más precisos y son más fiables en cuanto a la convergencia de los parámetros estimados a los parámetros reales del proceso.

A la hora de realizar una identificación no se suele ser de una única repetición sino que se suele seguir el siguiente proceso(Figura 7.11.0.7):

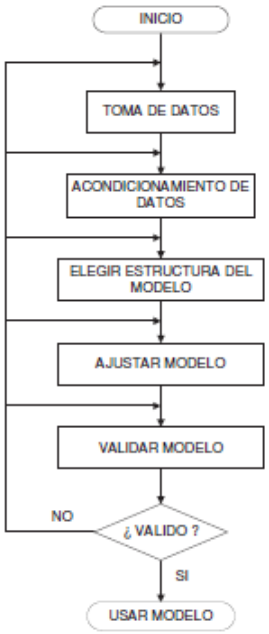


Figura 7.11.0.7 – Proceso a seguir a la hora de identificar un sistema

7.11.1. Identificación mediante mínimos cuadrados recursivos (RLS)

Uno de los métodos más utilizados para realizar la identificación es el de los mínimos cuadrados recursivos, conocido como RLS. Este método permite la identificación en tiempo real de modelos con el único requisito de que estos sean lineales en los parámetros.

El método en línea de RLS consiste en:

1. Dar valores iniciales a la matriz PN y al vector de parámetros θ .
2. En cada instante k se procederá:
   - Leer los valores de y(k) y u(k).
   - Formar el vector regresor FI(k) según la expresión

$$FI = [-y(2); -y(1); u(2)];$$

   - Calcular Ga(k) (ganancia de adaptación según la expresión

$$ga = lamda + FI' * PN_1 * FI$$

   - Calcular PN(k) (matriz de covarianza) mediante:

$$PN = (PN_1 - (PN_1 * (FI * FI') * PN_1)/ga)/lamda$$

   - Calcula $\theta(k)$ (vector de parámetros) :

$$Teta = Teta_1 + (PN_1 * FI * (y(3) - FI' * Teta_1))/ga$$

   La implementacíon se llevaría a cabo segun el esquema que se muestra en la Figura 7.11.1.1.

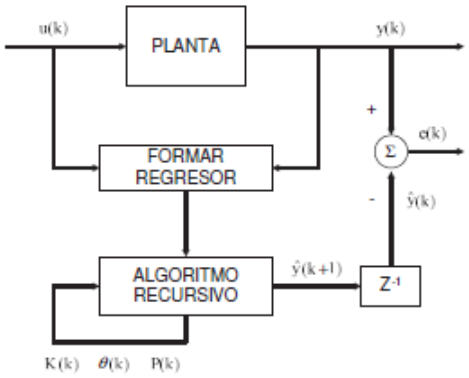


Figura 7.11.1.1 – Esquema identificación mediante RLS

EL RIS implementado permite obtener una función de transferencia de 2 polos. Se podría realizar otro tipo de rls como uno en el que se considere una función de transferencia de 2 polos y 1 polo. La principal diferencia entre ambas es que la función de transferencia de 2 polos y 1 cero indicaría que se trata de un sistema más rápido del considerado con la función de 2 polos. Finalmente se eligío la primera opción por considerar que se acerca lo suficiente al sistema real, por tanto es una aproximación buena del sistema. Una forma de comprobar la validez del método utilizado se puede comprobar mediante la toolbox que dispone Matlab para la indentificación. Esta herramienta se llama ident y se accede a ella ecribiendo su nombre en la ventana de comandos de Matlab. A la hora de utilizarla habra que utilizar los datos de la señal de entrada, los datos de la salida y la función de transferencia obtenida. Con estos parámetros se obtiene si la función es válida. Se considerará cuando presente un porcentaje superior del 90 %.

7.12. GUI (MATLAB)

La aplicación GUI de matlab permite crear interfaces gráficas donde mostrar la variación de las señales, asi como realizar modificaciones en las mismas. Estas interfaces están formadas por dos archivos: en primer lugar se generará la pantalla que se mostrará al usuario siendo su extensión .fig. Será en este archivo donde se guarden las características de los elementos introducidos, botones, gráficas, menús, barras de deslizamiento. Posteriormente y tras guardar este archivo se crea automáticamente otro con extensión .m. Este archivo será el que se ejecutará una vez terminado el programa. Aquí será donde se realice toda la programación de los elementos introducidos en la pantalla gráfica. Cada elemento introducido contará con 2 funciones en dicho archivo siendo utilizada para la programación la funcion callback. Será en es-

tas funciones donde se indicarán las instrucciones que se deben ejecutar al variar un elemento. Otra función que se genera para cada elemento es createfcn donde se especifican algunas características del elemento(no empleada).

Asimismo se generan funciones propias de la GUI para poder llevarse esta a cabo:

$functionvarargout = Manual(varargin) - >$ permite generar la GUI

$functionManual_{O}peningFcn(hObject, eventdata, handles, varargin) - >$Esta función se ejecutará unicamente después de ejecutar la GUI, es una función de inicialización de las variables introducidas en la misma.

$functionvarargout = Manual_{O}utputFcn(hObject, eventdata, handles) - >$ Será en esta función donde se ejecutará el progrma principal de la interfaz gráfica. Aqui se relizará toda la programación, excepto aquella que se debe realizar la pulsar un botón o algun elemento.

A la hora de querer mostrar o leer un valor de los elementos gráficos se utilizarán dos instrucciones básicas. La instrucción SET permite modificar las características de un handle, permite mostrar el valor de una variable en algún elemento de la GUI. La instrucción GET permite leer un valor introducido en los elementos gráficos y asignárselo a un variable del programa. Estas instrucciones se han de utilizar con las Tags (variables intermedias entre la programación y la interfaz gráfica).

Un aspecto a tener en cuenta a la hora de realizar la programación es tener cuidado con las variables. Por defecto las variables son locales, propias de cada función, y no se reconocen por el resto de funciones. Para poder interactuar entre callbacks y poder escribir y leer las variables que se relacionan con la parte gráfica será necesario utilizar la instrucción handles, así una variable çonsigna"se debe usar de la forma handles.consigna. Además para utilizar variables en varias funciones es necesario declararlas como globales, para ello se han de declarar globales en el inicio de cada función en las que son utilizadas.

Capítulo 8

Resultados finales

8.1. Interfaces Gráficas

La necesidad de realizar un control de una planta de laboratorio que no siempre puede ser visualizada a la hora de proceder a su control, hace necesario la realización de una interfaz gráfica. La interfaz permitirá apreciar en todo momento el estado de la planta así como los parámetros que se están a emplear en cada momento para su control. La interfaz permitirá variar los valores de las variables de forma fácil e intuitiva. Por estos motivos se optó por crear 4 interfaces gráficas mediante la herramienta GUI que incluye el software MATLAB. Se realizaron 4 interfaces para separar cada modo de funcionamiento en una pantalla y ser de esto modo mas entendible por el usuario.

En estas interfaces se muestran todas las variables de la planta al mismo tiempo que se visualiza el valor de los sensores así como la identificación de la misma mediante el método RLS. El usuario puede variar los valores de algunos de los parámetros como es el caso de los parámetros del PID, el nivel del depósito deseado, o el grado de apertura de las válvulas. Otro de los motivos para la creación de estas pantallas es la posibilidad de poder ver en gráficas como varían las variables a lo largo del tiempo.

Las pantallas están configuradas de la forma que se describe en la Figura 8.1.0.1:

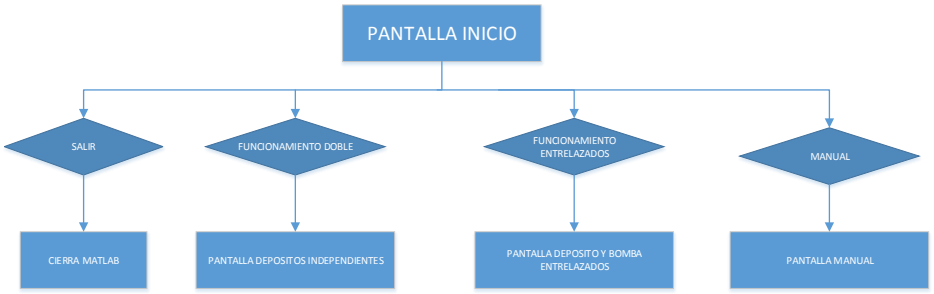


Figura 8.1.0.1 – Esquema interfaces gráficas

En primer lugar, al ejecutar el programa se presenta una pantalla en la cuál se permite al usuario elegir el modo de funcionamiento deseado a la vez que se le permite al usuario salir del programa. Posteriormente y según la elección del usuario se le envía a una de las 3 pantallas restantes, teniendo ya en estas la posibilidad de controlar y monitorizar la planta. Después de acceder a uno de los funcionamientos el usuario puede regresar a la pantalla inicial y desde aquí utilizar otro de los modos(Figura 8.1.0.2).

Figura 8.1.0.2 – Pantalla de inicio

Antes de mostrar las pantallas a las que se accederá se relizará una pequeña explicación de las mismas. Todas las pantallas siguen el mismo patrón. Las pantallas esta dividas en 4 partes fundamentales. Por un lado existe un panel para cada uno de los lados de la planta. Existe un panel para el depósito y bomba derecha y otro panel similar para el lado izquierod.Estos paneles tendrán incorporados el panel de indentificacón de la función de transferencia del sistema. Otro panel permite indicar los parámetros relativos al PID y el modo de control deseado. Por último existe un panel en el que se pueden realizar las perturbaciones al sistema. Con esta división se pretende ayudar al usuario en el uso del sistema. Cada pantalla contará con un botón extra que permite el regreso a la pantalla inicial. En el caso de los depósitos entrelazos no existen dos depósitos que controlar por lo que solo existirá un panel de control de la consigna. Además en este caso, la bomba del lado derecho actuará como perturbación.

En caso de seleccionar la opción de de funcionamiento doble el usuario se encontrará con la pantalla mostrada en la Figura 8.1.0.3.

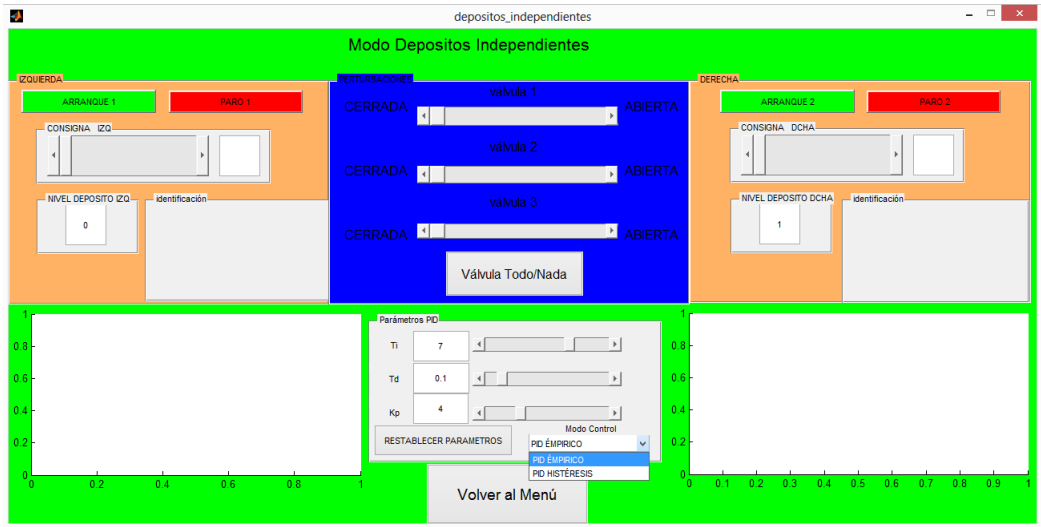


Figura 8.1.0.3 – Pantalla de funcionamiento independiente

Si por el contrario el usuario decide realizar una configuración de depósito y bomba entrelazados (bomba izquierda con depósito de la derecha) la pantalla a la que accederá será la de la Figura 8.1.0.4.

Por último si el usuario desea hacer un control manual de las bombas y ver nivel de los depósitos este accederá a la pantalla de la Figura 8.1.0.5.

Figura 8.1.0.4 – Pantalla de funcionamiento de bomba y depósito entrelazados

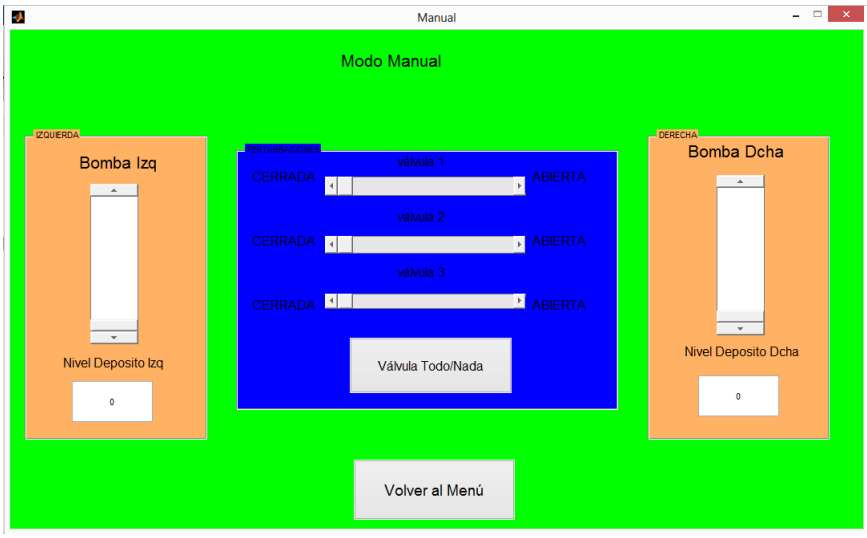


Figura 8.1.0.5 – Pantalla de funcionamiento manual

8.2. Modos de Funcionamiento

La existencia de varias bombas y depósitos proporciona a la planta diferentes formas de ser utilizada. En este proyecto se optó por realizar 3 modos de funcionamiento independientes cada uno con su pantalla gráfica propia.

8.2.1. Manual

El modo manual permite al usuario realizar un control manual sobre la potencia de las bombas, pudiendo seleccionar desde la bomba apagada hasta el 100 % de la potencia de la misma. Al mismo tiempo visualiza el nivel del depósito de forma

numérica. A la hora de modificar el sistema el usuario puede abrir o cerrar las distintas válvulas pudiendo así ver la respuesta del sistema ante perturbaciones. Este modo no permite un control de la planta sino una monitorización de las variables. Será el usuario el que realice el control del sistema. Su esquema será el de la Figura 8.2.1.1

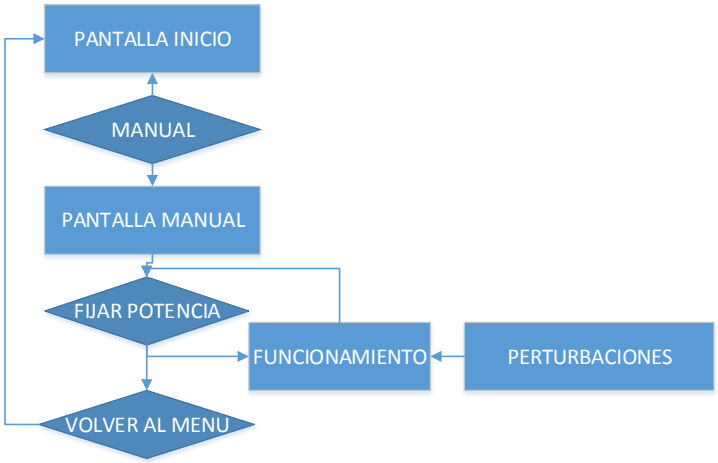


Figura 8.2.1.1 – Esquema modo manual

8.2.2. Depósitos independientes

El modo de depósitos independientes permite realizar dos controles independientes (control de los depósitos por separado).

En primer lugar se puede realizar el control de un único depósito mediante unos parámetros PID ya establecidos. Así mismo el usuario puede optar por cambiar estos valores y ver las modificaciones en la respuesta del sistema.

Otra posibilidad con un depósito es un un control mediante un PID con unos parámetros obtenidos mediante el método Relé explicado anteriormente.

Durante el funcionamiento el usuario puede modificar el estado de las válvulas o de la consigna para ver el comportamiento del sistema ante perturbaciones.

Cuando el programa este en ejecución el usuario podrá ver el comportamiento del sistema mediante una gráfica en las que se visualiza de color azul la consigna, de color rojo la potencia de la bomba y de color verde el nivel en cada instante del depósito.Así mismo se podrá ver la función de transferencia en Z del sistema obtenida en cada instante en el panel correspondiente.

En caso de querer funcionar con los dos depósitos de forma independientes el usuario debe cerrar las válvulas centrales (válvula todo o nada y la válvula proporcional 1) quedando así únicamente las válvulas 1 y 3 como posibles perturbaciones.En caso de emplear las válvulas centrales el usuario debe comprender que se pasaría a trabajar con un único depósito unido por una tubería y que dicha perturbación afecta a ámbos depósitos. El control que se realizará sobre los depósitos será el mismo, si se selecciona el método relé se usarán los mismos parámetros en los dos depósitos. En caso de seleccionar el modo empírico y querer variar los valor de Td, Ti, Kp los cambios influirán en los dos depósitos. Aunque los depósitos son independientes el modo de control y el valor de los parámetros no lo son.

Cuando se utilicen los dos depósitos a la vez cada una de las gráficas mostrará la evolución de las señales de cada uno de los depósitos. Cada depósito tendrá su identificación independiente no teniendo que ser estas iguales aunque los sistemas sean idénticos. Hay factores externos que influyen en dicho cálculo. El funcionamiento se regirá por el siguiente esquema (Figura 8.2.2.1).

8.2.3. Depósito y bomba entrelazados

Este modo de funcionamiento obliga a tener al menos la válvula proporcional 1 o la válvula todo o nada abiertas en todo momento, en caso contrario no se podrá realizar el control al no existir transvase de agua al depósito de la derecha. Anteriormente ya se indico que se lo óptimo sería mantener ambas abiertas. En este modo funcionará la

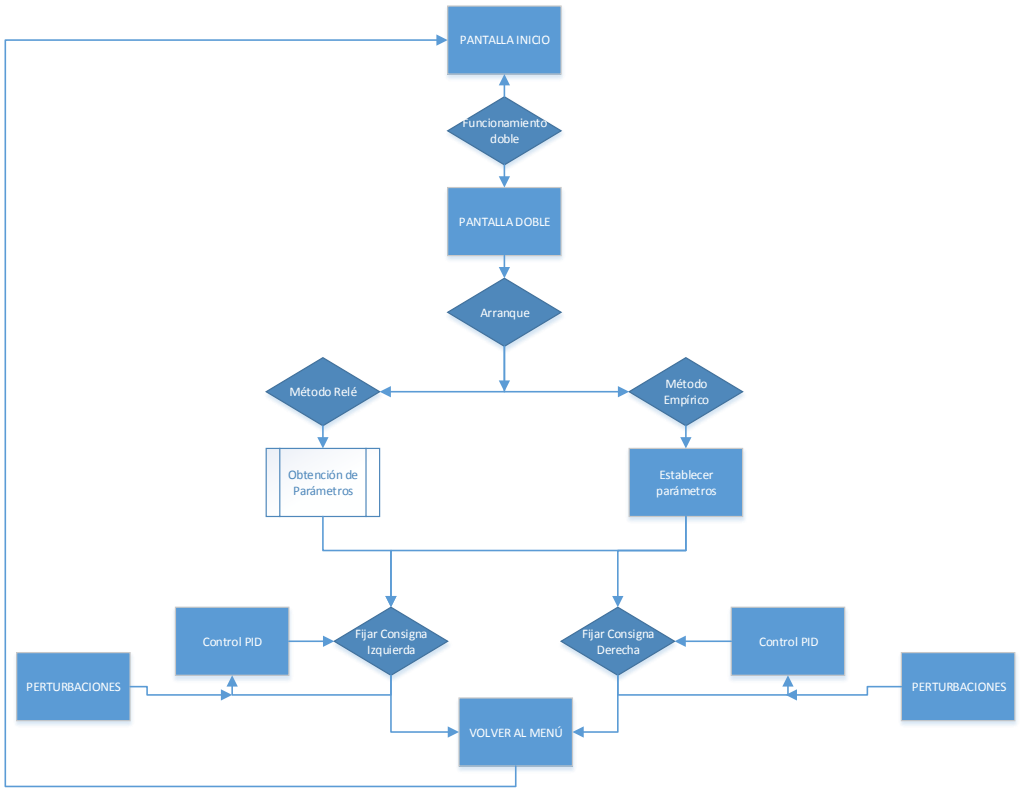


Figura 8.2.2.1 – Esquema depósitos independientes

bomba de la izquierda y se realizará la medida en el depósito de la derecha. En este modo la bomba de la derecha funcionará como perturbación dado que puede llenar o vaciar el depósito de la derecha. Se podría realizar la configuración contraria pero se optó por implementar esta.

A la hora de querer realizar perturbaciones al sistema (válvulas laterales) hay que tener en cuenta que la unión entre depósitos es mediante una tubería de 25mm de sección, la misma que tienen las perturbaciones, por tanto el rango en el que se deben variar las perturbaciones no debe ser en todo su rango(rango óptimo mencionado en el apartado de requisitos físicos). Al mismo tiempo se observará posteriormente que el sistema posee cierta inercia debido al retardo producido por las pérdidas de la tubería. Además existirá el retardo debido a que se deberán llenar prácticamente los dos depósitos al mismo nivel que el de la consigna. Esto se debe a que los depósitos funcionan como vasos comunicantes. Estos motivos producen una gran sobreoscilación en los momentos de puesta en marcha.

Otra cosa a tener en cuenta es que no posible alcanzar un nivel del 100 % dado a

las limitaciones físicas. Se recomienda un rango de funcionamiento del 15 % al 50 % del depósito para tener un funcionamiento óptimo. En el resto del rango no se asegura un correcto funcionamiento del control automático. Se presenta a continuación un esquema del modo de funcionamiento(Figura 8.2.3.1).

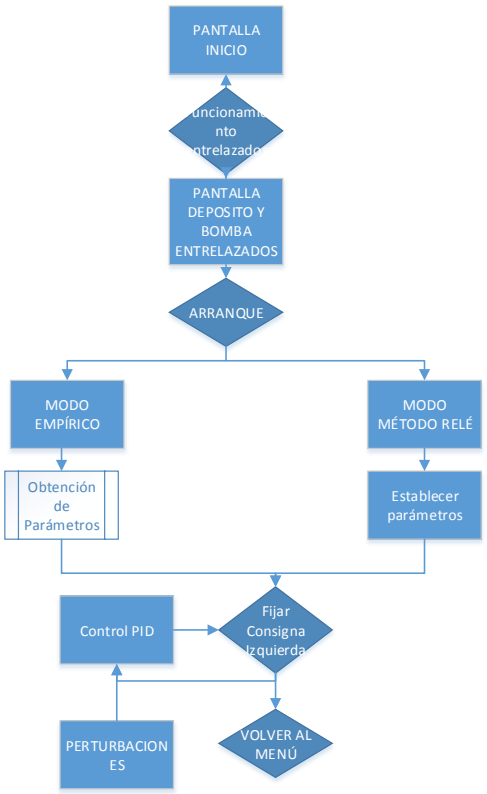


Figura 8.2.3.1 – Esquema depósito y bomba entrelazados

8.3. Puesta en marcha

Tras seguir los pasos descriptos anteriormente y tras el conexionado de todas las variables se pone en marcha la planta y se comprueba su funcionamiento en los diferentes modos. Tras las pruebas se comentarán las limitaciones físicas de la planta vistas en su funcionamiento.

En primer lugar se accederá a Matlab y se abrirá el archivo *pantalla*$_i$*nicio.m* entrando así en el código de la pantalla general.A continuación ejecutaremos dicho código pulsando el botón RUN del menú de Matlab. En este punto se abrirá la pantalla inicial donde podremos seleccionar el modo de funcionamiento. A la hora de pulsar

en uno de los modos de funcionamiento se realizará la conexión con la placa Arduino MEGA(Figura 8.3.0.2).

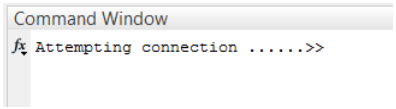


Figura 8.3.0.2 – Conexión con Arduino Mega

Si la conexión es válida aparecerá el siguiente mensaje (conexión satisfactoria) en la ventana de comandos de Matlab(Figura 8.3.0.3).

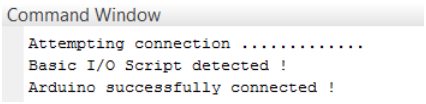


Figura 8.3.0.3 – Conexión con Arduino Mega satisfactoria

8.3.1. Manual

A continuación se presenta el modo manual en funcionamiento. En este caso encontramos la válvula 1 cerrada y las válvulas 2 y 3 abiertas, además la válvula todo o nada se encuentra abierta (indicado con el panel indicador).Este control depende del usuario por tanto cada usuario puede realizar el control que desee. Pueden funcionar una, otra o las dos bombas a la vez.(figura 8.3.1.1)

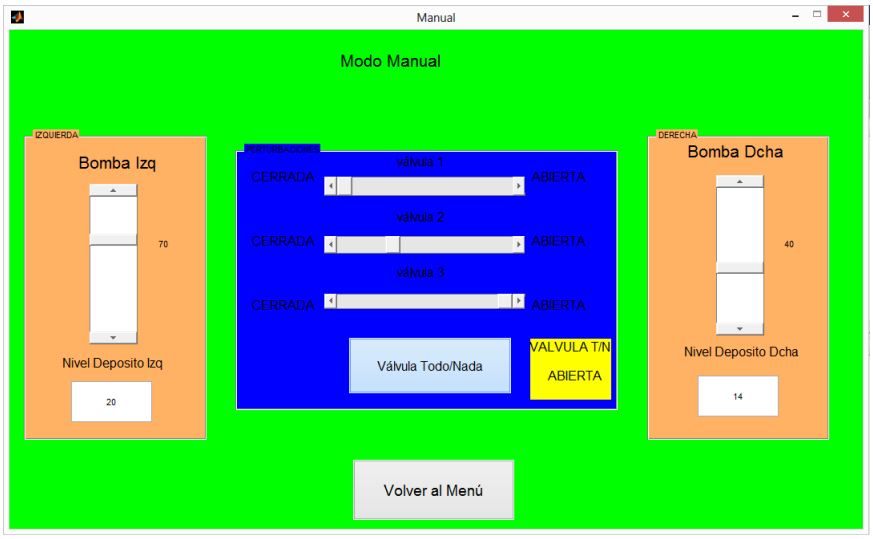


Figura 8.3.1.1 – Funcionamiento modo manual

8.3.2. Depósitos independientes

Los depósitos en modo independiente permiten diferentes controles y situaciones. Las siguientes figuras decribirán un poco cada uno de los estados en que se puede poner a la planta, viéndose la respuesta del sistema para cada una de ellas y cuál es la identificación realizada para cada una de ellas.

En primer lugar se muestra el funcionamiento del sistema en un único depósito.

Antes de comenzar se realizará el cálculo de los parámetros del PID mediante el método relé. Para ello se pulsa el botón de arranque de la bomba (se comprobara que no esta pulsado el botón de paro) de la izquierda y posteriormente se selecciona el modo método relé. En este momento se abrirá una nueva pantalla en la que se verá el proceso llevado a cabo, el cuál se muestra a continuación(figura 8.3.2.1).

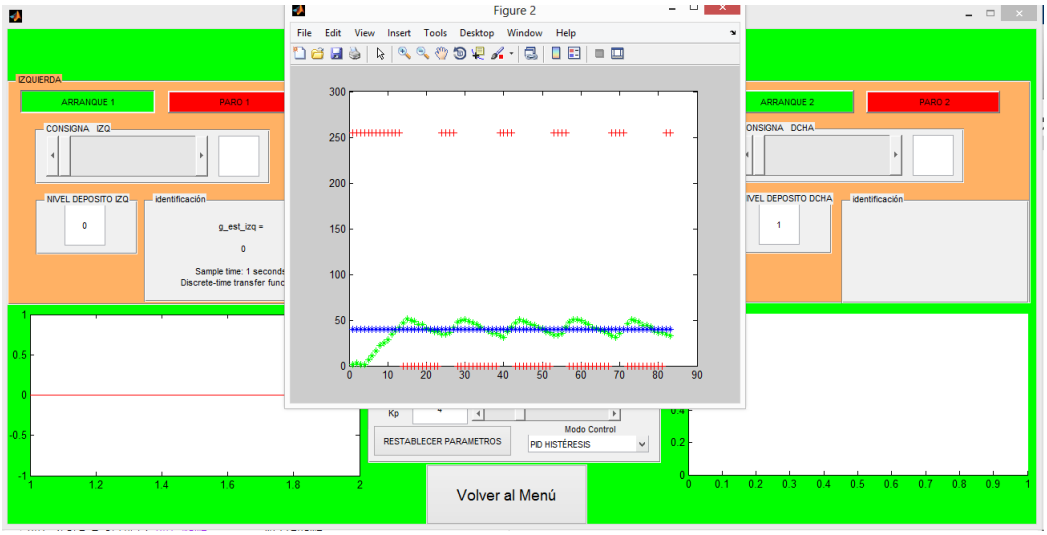


Figura 8.3.2.1 – Obtención de parámetros mediante método relé

Tras este proceso se obtienen los siguientes valor para las constantes Td,Ti y Kp(figura 8.3.2.2). Estos parámetros serán los mismos para ambos depósitos.

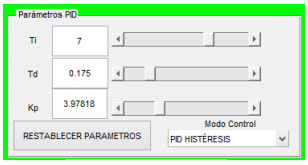


Figura 8.3.2.2 – Parámetros PID obtenidos mediante método relé

Con estos parámetros el sistema responderá de la siguiente manera(figura 8.3.2.3):

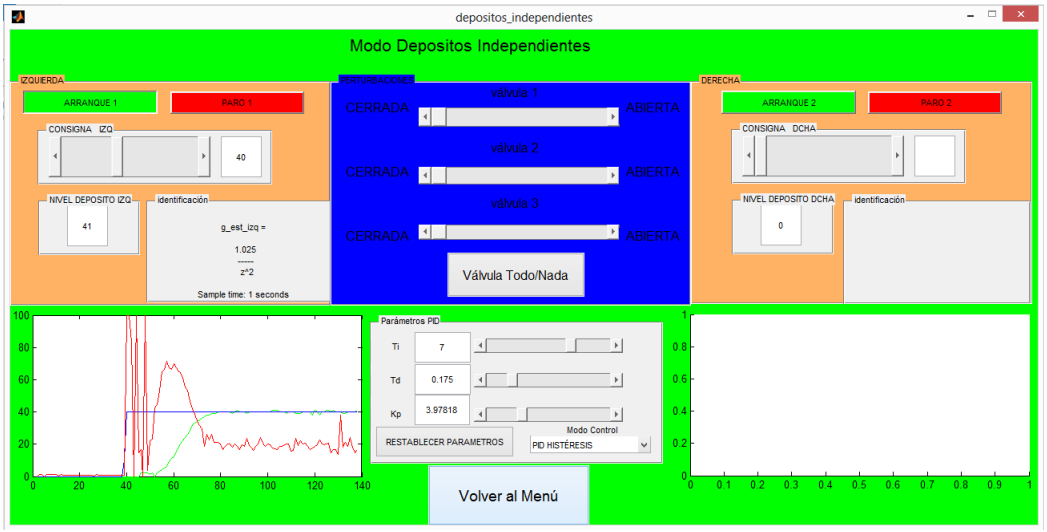


Figura 8.3.2.3 – Funcionamiento 1 depósito mediante método relé

Posteriormente se cambia el valor de la consigna y se comprueba su respuesta al cambio(figura 8.3.2.4).

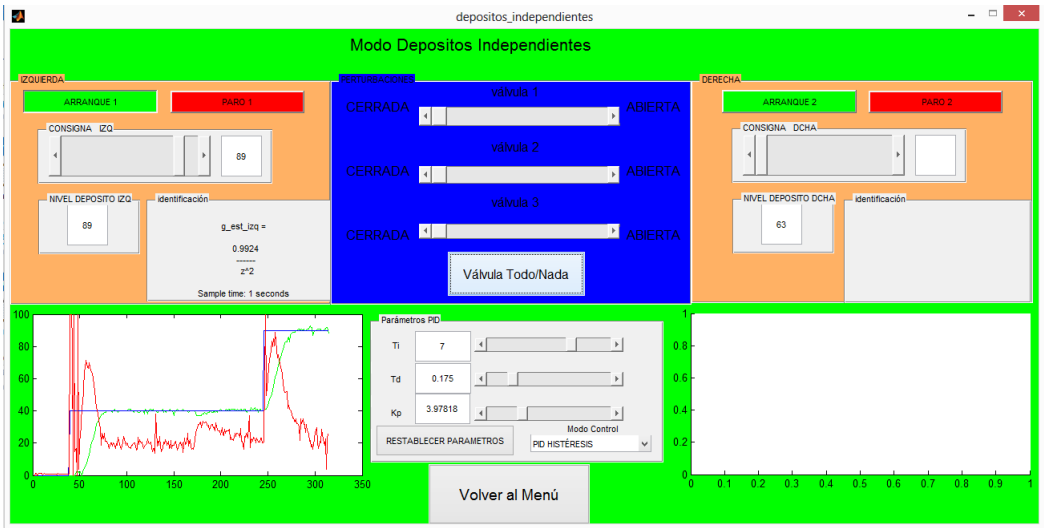


Figura 8.3.2.4 – Funcionamiento 1 depósito mediante método relé con cambio de consigna

Se comprueba que el tiempo de respuesta del sitema no es elevado y asumible para el sistema que se trata.

Otra cosa a tener en cuenta es saber cuál es la respuesta del sistema ante una perturbación. Por este motivo se realiza una prueba modificando el valor de alguna dde las válvulas siendo esta la respuesta(figura 8.3.2.5) :

Por último en el método relé se probo el funcionamiento de dos los depósitos a la vez(figura 8.3.2.6).

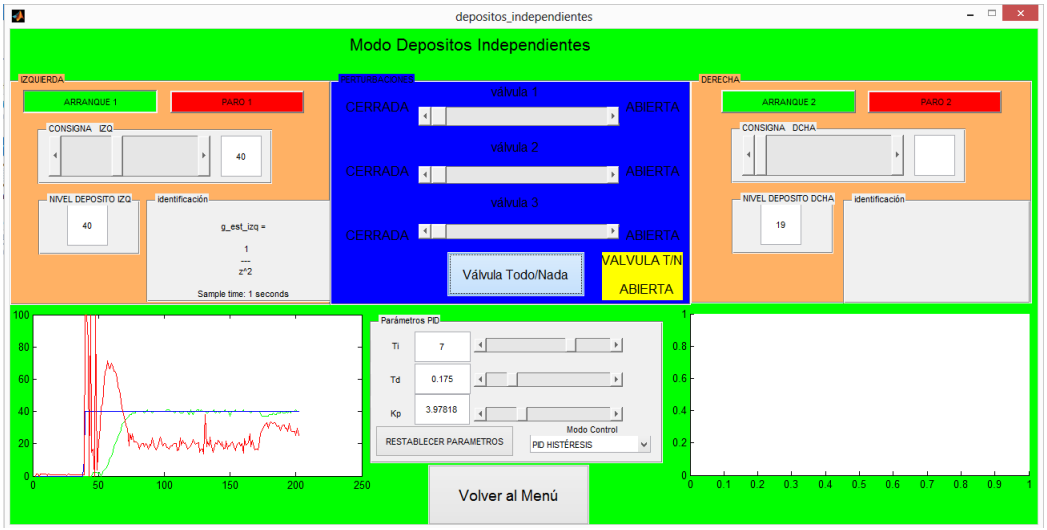


Figura 8.3.2.5 – Funcionamiento 1 depósito mediante método relé ante perturbación

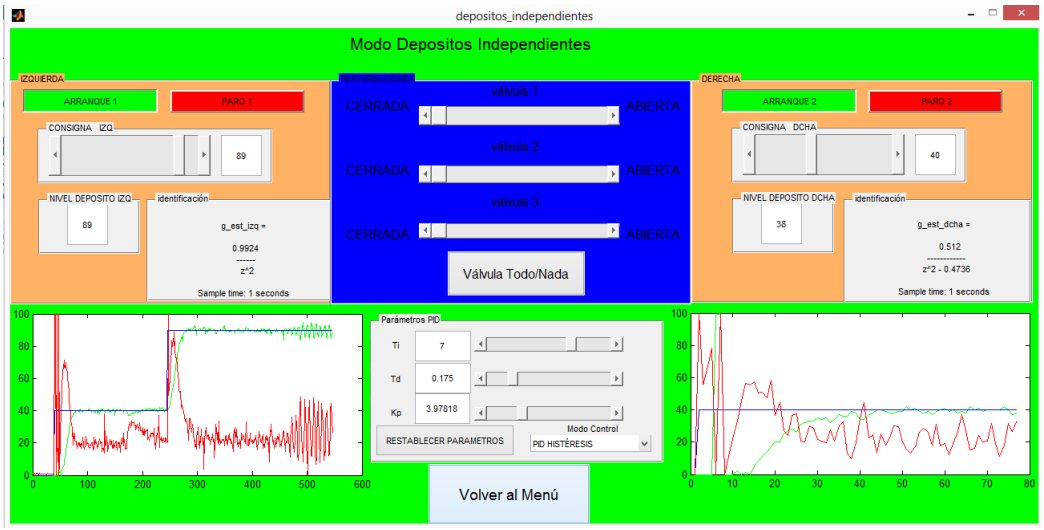


Figura 8.3.2.6 – Funcionamiento 2 depósitos mediante método relé

Como se comento anteriormente a veces las señales leidas por el sensor de nivel no son correctas (existencia de aire en las tuberías).Este problema no es hábitual de la planta, suele suceder en el primer arranque de la planta tras estar parada o cuando se llego a vaciar un depósito mediante la bomba. Se produce únicamente en estos casos porque es aqui cuando las tuberías de llenado se llenan de aire. Esto se puede comprobar en la siguiente figura(figura 8.3.2.7). Para ello debemos fijarnos en la señal de color verde de la gráfica de la derecha.

Tras el método de obtención de parámetros mediante la histéresis se comprueba el funcionamiento del PID mediante los parámetros empíricos. Los valores de estos parámetros se muestran en la siguiente figura(figura 8.3.2.8) (valores prefijados). En este método se podrían realizar infinitas pruebas variando únicamnte los parámetros

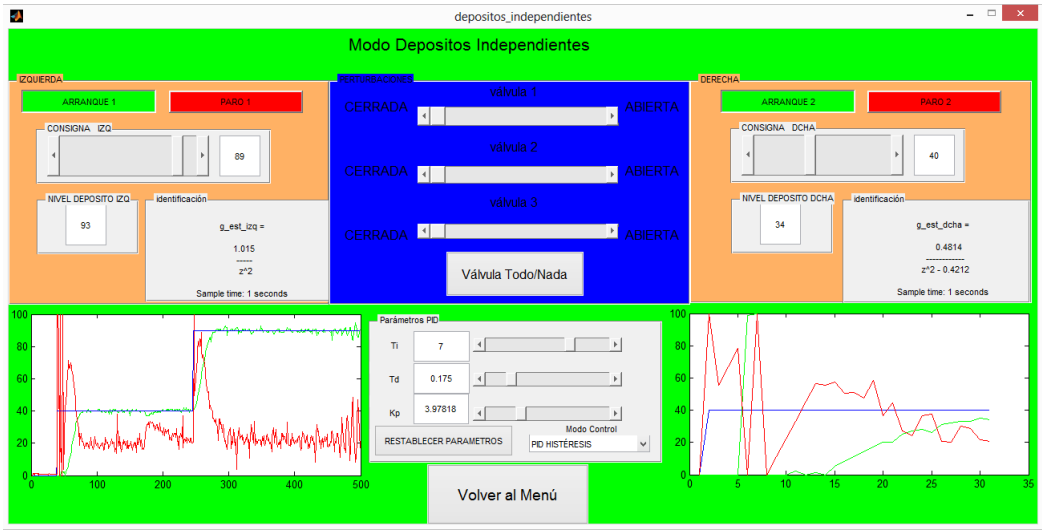


Figura 8.3.2.7 – Problemas causados por la existencia de burbujas

del regulador PId.

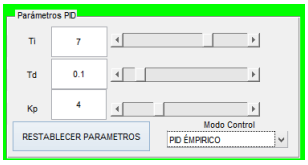


Figura 8.3.2.8 – Valor de los parámetros en el modo empírico

Con estos parámetros se obtiene una respuesta del sistema para 1 depósito(figura 8.3.2.9):

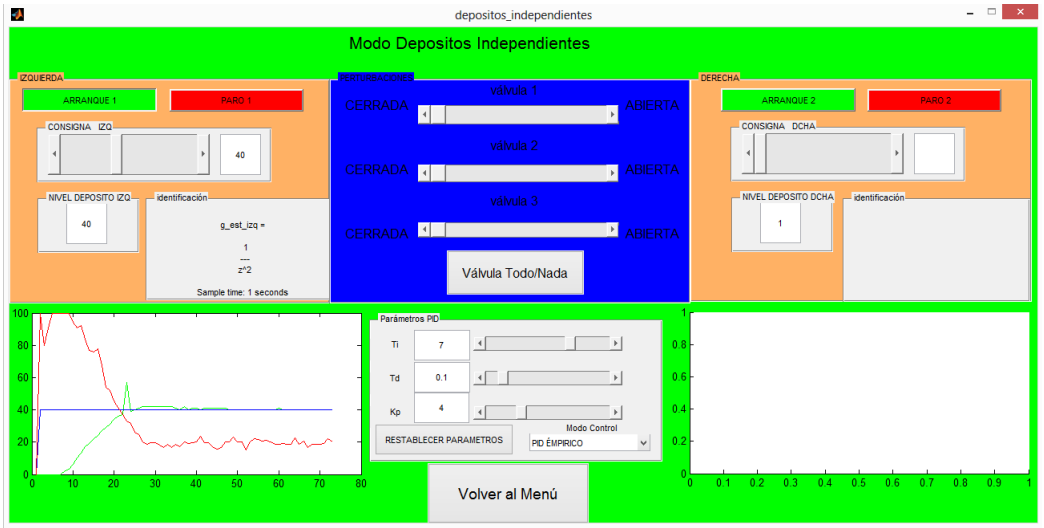


Figura 8.3.2.9 – Funcionamiento de 1 depósito en el modo empírico

Sí se produce un cambio de consigna el sistema responde de la siguiente manera

(figura 8.3.2.10):

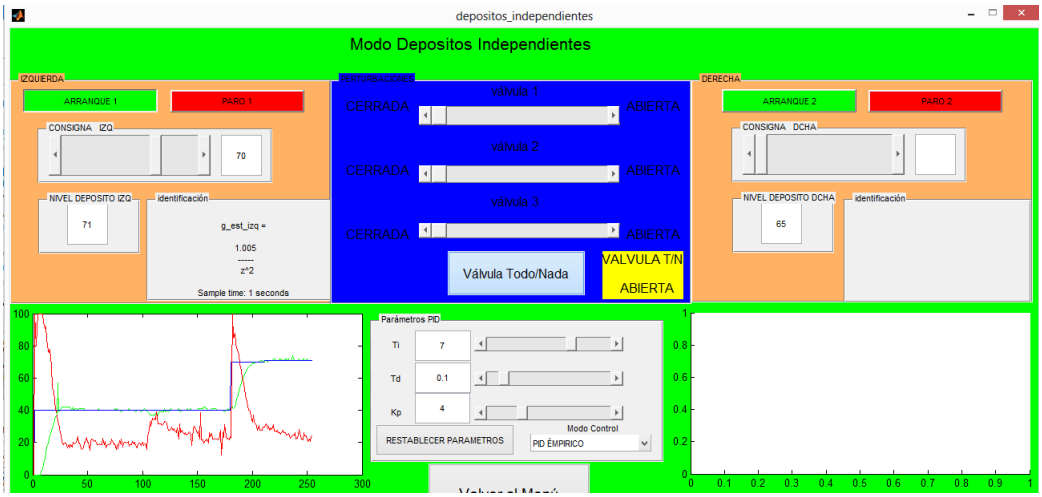


Figura 8.3.2.10 – Funcionamiento de 1 depósito en el modo empírico ante un cambio de consigna

En cambio si se produce una perturbación el sistema responde así(figura 8.3.2.11):

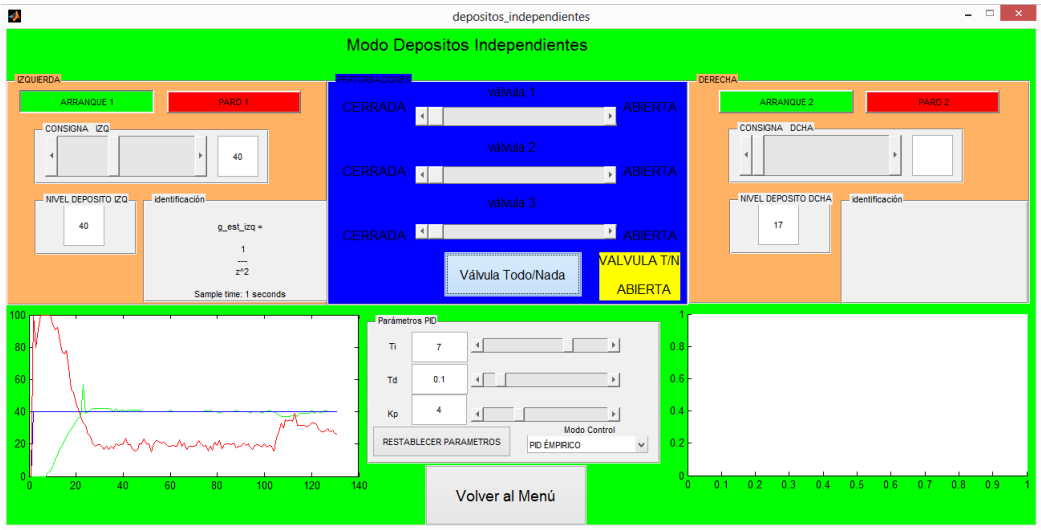


Figura 8.3.2.11 – Funcionamiento de 1 depósito en el modo empírico ante una perturbación

Las imágenes demuestran que la respuesta del sistema es buena tanto ante perturbaciones como a los cambios de consigna. El error cometido como máximo a la hora de conseguir el nivel indicado es del 2 % siendo este error aceptable para el correcto funcionamiento de la planta. También se ve que los sensores en alguna ocasión indican un valor de nivel de agua que no es el correcto y como este influye en el sistema.

8.3.3. Depósito y bomba entrelazados

El funcionamiento de la bomba y depósito entrelados resulta complejo. Anteriormente ya se indico que es un sistema con una gran retardo. Este retardo produce grandes oscilaciones sobre todo al comienzo del funcionamiento, mientrás que al final se produce una estabilización del sistema. El error cometido en este caso es superior al modo anterior no siendo superior al 5 % después de un tiempo de estalibización(figura 8.3.3.1).

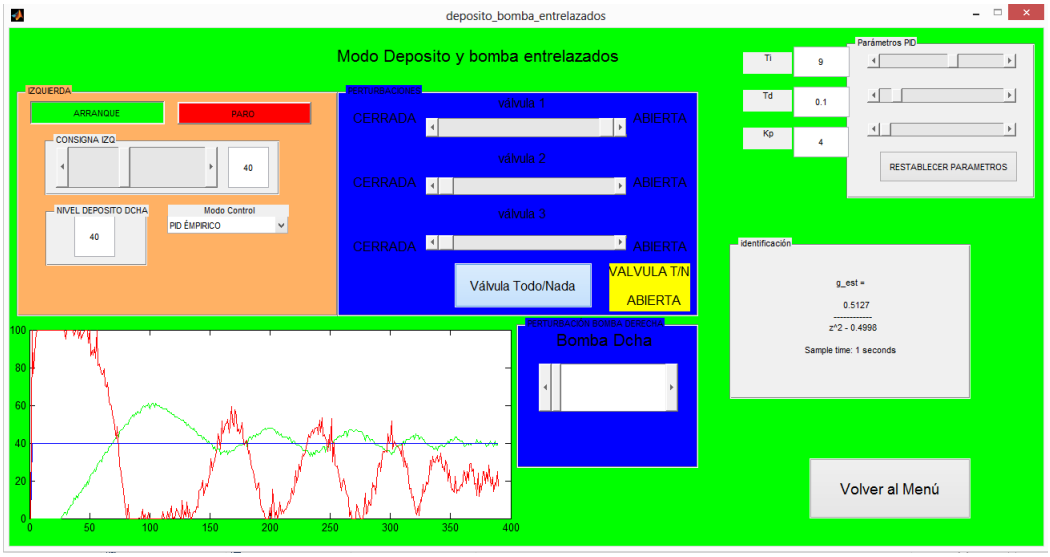


Figura 8.3.3.1 – Estabilización del sitema de depósito y bomba entrelazados

Además se puede comprobar en la imágen anterior que los valores configurados para el PID son diferentes a los del modo de depósitos independientes, al tratarse como ya se mencionó de sistemas diferentes. En este caso se obtienen unos parámetros mediante el método rele de((figura 8.3.3.2):

Figura 8.3.3.2 – Parámetros método relé para bomba y depósito entrelazados

El funcionamiento del sisteme en el modo entrelazados es el suiguiente (figura 8.3.3.2)

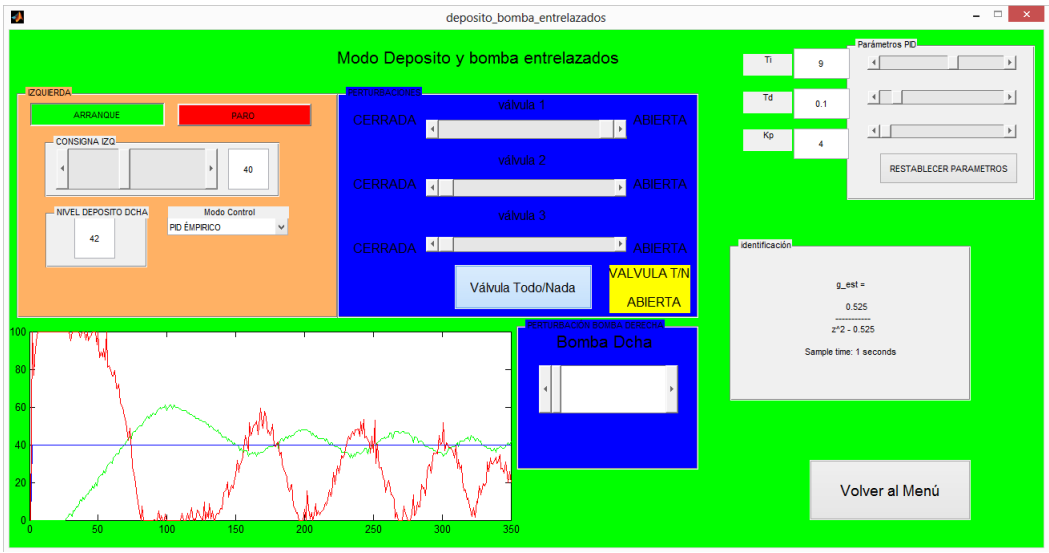


Figura 8.3.3.3 – Funcionamiento del sistema para bomba y depósito entrelazados

ANEXOS

TÍTULO: **ESTRATEGIAS DE CONTROL PARA PLANTA DE LABORATORIO CON RETARDO**

ANEXOS

PETICIONARIO: **ESCOLA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBRERO, S/N

15405 - FERROL

FECHA: SEPTIEMBRE DE 2014

AUTOR: EL ALUMNO

Fdo.: RUBÉN VALIÑO DÍAZ

Capítulo 9

Documentación de partida

ESCUELA UNIVERSITARIA POLITÉCNICA

ASIGNACIÓN DE TRABAJO FIN DE GRADO

En virtud de la solicitud efectuada por:
En virtud da solicitude efectuada por:

APELLIDOS, NOMBRE: *Valiño Díaz,Rubén*
APELIDOS E NOME:

DNI: **Fecha de Solicitud:** FEB2014
DNI: *Fecha de Solicitude:*

Alumno de esta escuela en la titulación de Grado en Ingeniería en Electrónica Industrial y Automática, se le comunica que la Comisión de Proyectos ha decidido asignarle el siguiente Trabajo Fin de Grado:

O alumno de esta escola na titulación de Grado en Enxeñería en Electrónica Industrial e Automática, comunícaselle que a Comisión de Proxectos ha decidido asignarlle o seguinte Traballo Fin de Grado:

Título T.F.G:Estrategias de control para Planta de Laboratorio con Retardo

Número TFG: *770G01A53*

TUTOR:(Titor) *Calvo Rolle,Jose Luis*

COTUTOR/CODIRECTOR: *José Luis Casteleiro Roca*

La descripción y objetivos del Trabajo son los que figuran en el reverso de este documento*:*

A descrición e obxectivos do proxecto son os que figuran no reverso deste documento.

Ferrol a Viernes, 7 de Marzo del 2014

Retirei o meu Traballo Fin de Grado o dia _____ de _____ do ano _____

Fdo: Valiño Díaz,Rubén

DESCRIPCIÓN Y OBJETIVO:Objeto
El presente proyecto versará sobre las diferentes acciones necesarias para poner en funcionamiento una Planta de Laboratorio y, la implementación de un programa en Matlab para la comprobar el funcionamiento de la misma.
También se creará un programa para el control automático del nivel con los dos depósitos interconectados realizando la aportación por uno y el control en el siguiente.

Alcance
?Configuración de los variadores de las bombas.
?Configuración de los sensores de nivel.
?Comprobación de las válvulas de paso.
?Comprobación de los sensores de seguridad.
?Implementación de un programa de control manual de todas las variables de entrada y visualización de las salidas.
?Introducción de un regulador PID para el control automático del nivel con los dos depósitos interconectados.

Capítulo 10

Código

10.1. Función PID

```
function [ output_value,Error,Control_signal ] = PID_PROPIO( kp
    ,Ti,Td,input_voltage,Error,Control_signal,nivel_deseado );
%Realiza el controlador PID del sistema
wait_time=0;
T=0.5;

    start_time=toc; %Almacena el tiempo de inicio de la
       muestra
    %Instrucciones
    variable_anterior=input_voltage;
     if input_voltage<(variable_anterior -100)
       input_voltage=variable_anterior;
     end
    nivel_actual=((1023-input_voltage)*100)/1023; %Cálculo del
        nivel
    %Calculo de los errores
    Error(1)=Error(2);
    Error(2)=Error(3);
    Error(3)= nivel_deseado-nivel_actual;
    %Calculo de los pesos de influencia de cada error
    q0=kp*(1+(Td/T));
    q1=kp*(-1 +(T/Ti)-2*(Td/T));
    q2=kp*(Td/T);
```

```
    %Determinacion de las señales de control
    Control_signal(1)=Control_signal(2);
    Control_signal(2)=Control_signal(1)+q0*Error(3)+q1*Error(2)+
       q2*Error(1);
     if Control_signal(2)>100
         Control_signal(2)=100;
     end;
     if Control_signal(2)<0
         Control_signal(2)=0;
     end;
     output_voltage=Control_signal(2)/100; % tensión de salida
     output_value=round((output_voltage*255)) %Escalado de la
        salida

     %Instrucciones para comprobar que se cumplió el tiempo de
        muestreo
     spend_time=toc-start_time; %Comprobamos si el tiempo
        transcurrido
     %permite cumplir con el período de muestreo
     if spend_time>T
         disp('Tiempo de muestreo superado'); %Mostramos la
            causa de fallo

     else
         wait_time=T-spend_time; %Si el tiempo transcurrido es
            menor, se
         pause(wait_time); %esperará hasta completar T
     end

end
```

10.2. Función método-relé

```
function [ Kp,Td,Ti ] = metodo_rele(a,seleccion);
%Realiza la configuracion de los parámetros del PID mediante
   el método de
%relé
```

```
%Varialbes histeresis
amplitud=0;
nivel_deseado=40;
tc=0; %periodo de oscilacion sostenida
d=0; %variable auxiliar que se utilizara para conocer el hueco
    entre
%periodos
h=0;
a.pinMode(2,'output'); %bomba2
a.pinMode(4,'output'); %bomba2
aux=0;
t=[ 0 0 0];
wait_time=0;
kcr=0; %Ganancia proporicional critca
flag=0;
mayor=0;
menor=100;
input_voltage=0; %Inicialización del valor de entrada
output_value=0; %Inicialización del valor de salida
T=1; %Período de muestreo
valor_superior=nivel_deseado+5;
valor_inferior=nivel_deseado-5;
tic; %Activación del temporizador
 hold off ;
        while(tc==0)

    start_time=toc; %Almacena el tiempo de inicio de la
       muestra
    %Instrucciones
    d=d+1;
    variable_anterior=input_voltage;
    input_voltage=(a.analogRead(0)-a.analogRead(2));; %Tensión
        en la entrada


    %control para evitar lectura de datos erronea mediante una
        mala medida del sensor de ultrasonidos
```

```
if input_voltage<(variable_anterior -100)

    input_voltage=variable_anterior;
end

 nivel_actual=((1023-input_voltage)*100)/1023; %Cálculo del
     nivel en %
%Produzco la histeresis del sitema entre un valor superior y
    un valor inferior
  if nivel_actual<=valor_inferior
    output_value=255;
    flag=0;
    %selecciono el menor valor que toma la lectura leida
    if nivel_actual<menor & aux>0
        menor=nivel_actual;
    end
 end
    if nivel_actual>=valor_superior
    output_value=0;
    if flag==0
        flag=1;

    aux=aux+1; %variable auxiliar que se incrementa en cada
        periodo
    t(aux)=d;
    end
    if nivel_actual>mayor
        mayor=nivel_actual ;
    end;

    end

amplitud=mayor-menor;
if seleccion==1
 a.analogWrite(2,output_value); %Genera la tensión de
    salida
end
```

```
 if seleccion==0
 a.analogWrite(4,output_value); %Genera la tensión de
    salida
end
 h=valor_superior-valor_inferior;


     %Instrucciones para comprobar que se cumplió el tiempo
        de muestreo
 spend_time=toc-start_time; %Comprobamos si el tiempo
    transcurrido
 %permite cumplir con el período de muestreo
 if spend_time>T
     disp('Tiempo de muestreo superado'); %Mostramos la
        causa de fallo
     break; %Si el tiempo es mayor aborta la muestra
 else
     wait_time=T-spend_time; %Si el tiempo transcurrido es
        menor, se
     pause(wait_time); %esperará hasta completar T
 end

 if aux==6 %En el 6º periodo se toma el valor de tc
     tc=(t(6)-t(5))*T;
 end
 %Muestro el proceso de calculo de la amplitud y de la
    ganacia critica
  figure(2);
 plot(d,nivel_actual,'g*');
   hold on ;
 plot(d,nivel_deseado,'b*');
 plot(d,output_value,'r+');
     end
   %Cálculo de los parámetros del PID
 kcr=4*100/(pi*(amplitud^2-5^2)^(0.5));
 Kp=0.6*kcr;
 Ti=0.5*tc;
```

```
    Td=(0.125*tc)*0.1;
   hold off;

end
```

10.3. Función identificación RLS 2 polos

```
function [ g_est ] = rls_2polos(u,y )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here


%condiciones iniciales
a=0;
a1=0;
b=0;
lamda=0.99 %Factor de olvido
Teta_1=[-a;a1;b]; %vector parámetros
PN_1=[1000 0 0;0 1000 0;0 0 1000]; %matriz de covarianza


    FI=[-y(2); -y(1); u(2)]; %vector regresor
    ga= lamda+FI'*PN_1*FI;
    PN=(PN_1-(PN_1*(FI*FI')*PN_1)/ga)/lamda;
    Teta=Teta_1+(PN_1*FI*(y(3)-FI'*Teta_1))/ga;
    PN_1=PN;
    Teta_1=Teta;
    Teta1k(3)=Teta(1,1);
    Teta2k(3)=Teta(2,1);
    Teta3k(3)=Teta(3,1);

 a=Teta(1,1);
 a1=Teta(2,1);
 b=Teta(3,1);


num=b;
```

```
den=[1 a a1];
    %Se obtiene l función de trasferencia del sistema mediante
       RLS

g_est=tf(num,den,1)
```

end

Programa pantalla inicio:

```
function varargout = Pantalla_inicio(varargin)
%PANTALLA_INICIO MATLAB code for Pantalla_inicio.fig
%      PANTALLA_INICIO, by itself, creates a new
   PANTALLA_INICIO or raises the existing
%      singleton*.
%
%      H = PANTALLA_INICIO returns the handle to a new
   PANTALLA_INICIO or the handle to
%      the existing singleton*.
%
%      PANTALLA_INICIO('CALLBACK',hObject,eventData,handles
   ,...) calls the local
%      function named CALLBACK in PANTALLA_INICIO.M with the
   given input arguments.
%
%      PANTALLA_INICIO('Property','Value',...) creates a new
   PANTALLA_INICIO or raises the
%      existing singleton*.  Starting from the left, property
   value pairs are
%      applied to the GUI before Pantalla_inicio_OpeningFcn
   gets called.  An
%      unrecognized property name or invalid value makes
   property application
%      stop.  All inputs are passed to
   Pantalla_inicio_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI
   allows only one
```

```
%      instance to run (singleton)".
%
%See also: GUIDE, GUIDATA, GUIHANDLES

%Edit the above text to modify the response to help
   Pantalla_inicio

%Last Modified by GUIDE v2.5 14-Jul-2014 22:13:44

%Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
                     @Pantalla_inicio_OpeningFcn, ...
                   'gui_OutputFcn',  @Pantalla_inicio_OutputFcn
                     , ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin
       {:});
else
    gui_mainfcn(gui_State, varargin{:});
end
%End initialization code - DO NOT EDIT


%--- Executes just before Pantalla_inicio is made visible.
function Pantalla_inicio_OpeningFcn(hObject, eventdata, handles
   , varargin)

%This function has no output args, see OutputFcn.
```

```
%hObject    handle to figure
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)
%varargin   command line arguments to Pantalla_inicio (see
   VARARGIN)

imagen_planta=imread('planta_laboratorio.jpg');
image(imagen_planta);
axis off;
%Choose default command line output for Pantalla_inicio
handles.output = hObject;

%Update handles structure
guidata(hObject, handles);

%UIWAIT makes Pantalla_inicio wait for user response (see
   UIRESUME)
%uiwait(handles.figure1);


%--- Outputs from this function are returned to the command
   line.
function varargout = Pantalla_inicio_OutputFcn(hObject,
   eventdata, handles)
%varargout  cell array for returning output args (see
   VARARGOUT);
%hObject    handle to figure
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Get default command line output from handles structure
varargout{1} = handles.output;
```

```
%--- Executes on button press in funcionamiento_doble.
function funcionamiento_doble_Callback(hObject, eventdata,
    handles)
clear all; close all;clc; depositos_independientes;
%hObject    handle to funcionamiento_doble (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)


%--- Executes on button press in funcionamiento_entrelazado.
function funcionamiento_entrelazado_Callback(hObject, eventdata
    , handles)
clear all; close all;clc; deposito_bomba_entrelazados;
%hObject    handle to funcionamiento_entrelazado (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)

%--- Executes on button press in manual.
function manual_Callback(hObject, eventdata, handles)
clear all; close all;clc; Manual;
%hObject    handle to manual (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)


%--- Executes on button press in SALIR.
function SALIR_Callback(hObject, eventdata, handles)
clear all; close all;clc;
%hObject    handle to SALIR (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
```

```
%handles    structure with handles and user data (see GUIDATA)
```

10.4. Programa pantalla funcionamiento manual

```
function varargout = Manual(varargin)
%MANUAL MATLAB code for Manual.fig
%      MANUAL, by itself, creates a new MANUAL or raises the
   existing
%      singleton*.
%
%      H = MANUAL returns the handle to a new MANUAL or the
   handle to
%      the existing singleton*.
%
%      MANUAL('CALLBACK',hObject,eventData,handles,...) calls
   the local
%      function named CALLBACK in MANUAL.M with the given input
    arguments.
%
%      MANUAL('Property','Value',...) creates a new MANUAL or
   raises the
%      existing singleton*.  Starting from the left, property
   value pairs are
%      applied to the GUI before Manual_OpeningFcn gets called.
     An
%      unrecognized property name or invalid value makes
   property application
%      stop.  All inputs are passed to Manual_OpeningFcn via
   varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI
   allows only one
%      instance to run (singleton)".
%
%See also: GUIDE, GUIDATA, GUIHANDLES

%Edit the above text to modify the response to help Manual
```

```
%Last Modified by GUIDE v2.5 22-Aug-2014 11:58:45

%Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Manual_OpeningFcn, ...
                   'gui_OutputFcn',  @Manual_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin
       {:});
else
    gui_mainfcn(gui_State, varargin{:});
end
%End initialization code - DO NOT EDIT
end

%--- Executes just before Manual is made visible.
function Manual_OpeningFcn(hObject, eventdata, handles,
   varargin)

%This function has no output args, see OutputFcn.
%hObject    handle to figure
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)
%varargin   command line arguments to Manual (see VARARGIN)
a=arduino('COM3');
a.pinMode(2,'output'); %bomba1
a.pinMode(4,'output');%bomba2
```

```
a.pinMode(8,'output'); %valvula1
a.pinMode(10,'output'); %valvula2
a.pinMode(12,'output'); %valvula3
a.pinMode(24,'output'); %valvula todo/nada

handles.a=a;
handles.a.analogWrite(2,0);
handles.a.analogWrite(4,0);
set(handles.mos_valvulatn,'Visible','off');
%Choose default command line output for Manual
handles.output = hObject;

%Update handles structure
guidata(hObject, handles);

%UIWAIT makes Manual wait for user response (see UIRESUME)
%uiwait(handles.figure1);

end
%--- Outputs from this function are returned to the command
    line.

function varargout = Manual_OutputFcn(hObject, eventdata,
    handles)
%varargout  cell array for returning output args (see
    VARARGOUT);
%hObject    handle to figure
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)
%Get default command line output from handles structure
varargout{1} = handles.output;
%--- Executes on slider movement.
i=0;

while(1)
nivel_deposito_dcha=round(100-(((handles.a.analogRead(0)-
```

```
    handles.a.analogRead(2))*100)/1023));
set(handles.mos_dep_dcha,'string',fix(nivel_deposito_dcha));
nivel_deposito_izq= round(100-(((handles.a.analogRead(4)-
    handles.a.analogRead(6))*100)/1023));
set(handles.mos_dep_izq,'string',fix(nivel_deposito_izq));
if nivel_deposito_dcha>100
    handles.a.analogWrite(4,0);
end
if nivel_deposito_dcha<=95
pot_bomba_dcha=get(handles.bomba_dcha_pot,'value');
handles.a.analogWrite(4,round((pot_bomba_dcha*255)/100));
end

if nivel_deposito_izq>100
    handles.a.analogWrite(2,0);
end
if nivel_deposito_izq<=95
pot_bomba_izq=get(handles.bomba_izq_pot,'value');
handles.a.analogWrite(2,round((pot_bomba_izq*255)/100));
end
pause(0.5);

end
end


function bomba_izq_pot_Callback(hObject, eventdata, handles)

pot_bomba_izq=get(handles.bomba_izq_pot,'value');
handles.a.analogWrite(2,round((pot_bomba_izq*255)/100));
set(handles.mos_bomba_izq,'string',fix(pot_bomba_izq));


%hObject    handle to bomba_izq_pot (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider


%--- Executes during object creation, after setting all
   properties.
end
function bomba_izq_pot_CreateFcn(hObject, eventdata, handles)
% hObject    handle to bomba_izq_pot (see GCBO)
% eventdata  reserved - to be defined in a future version of
   MATLAB
% handles    empty - handles not created until after all
   CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
end

%--- Executes on slider movement.
function bomba_dcha_pot_Callback(hObject, eventdata, handles)

pot_bomba_dcha=get(handles.bomba_dcha_pot,'value');
handles.a.analogWrite(4,round((pot_bomba_dcha*255)/100));
set(handles.mos_bomba_dcha,'string',fix(pot_bomba_dcha));
% hObject    handle to bomba_izq_pot (see GCBO)
% eventdata  reserved - to be defined in a future version of
   MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider
```

```
end
%--- Executes during object creation, after setting all
    properties.
function bomba_dcha_pot_CreateFcn(hObject, eventdata, handles)
%hObject    handle to bomba_izq_pot (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    empty - handles not created until after all
    CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor',[.9 .9 .9]);
end
end


function mos_dep_izq_Callback(hObject, eventdata, handles)


end
%hObject    handle to mos_dep_izq (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'String') returns contents of mos_dep_izq
    as text
%        str2double(get(hObject,'String')) returns contents of
    mos_dep_izq as a double


%--- Executes during object creation, after setting all
    properties.
function mos_dep_izq_CreateFcn(hObject, eventdata, handles)
```

```
%hObject    handle to mos_dep_izq (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: edit controls usually have a white background on
   Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function mos_dep_dcha_Callback(hObject, eventdata, handles)

end
%hObject    handle to mos_dep_dcha (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'String') returns contents of mos_dep_dcha
    as text
%       str2double(get(hObject,'String')) returns contents of
   mos_dep_dcha as a double


%--- Executes during object creation, after setting all
   properties.
function mos_dep_dcha_CreateFcn(hObject, eventdata, handles)
%hObject    handle to mos_dep_dcha (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
```

```
    CreateFcns called

% Hint: edit controls usually have a white background on
    Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

end
%--- Executes on slider movement.
function pos_valvula1_Callback(hObject, eventdata, handles)
posicion_valvula1=get(handles.pos_valvula1,'value');
handles.a.analogWrite(12,round((posicion_valvula1*255)/100));



%hObject    handle to pos_valvula1 (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%       get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider

end
%--- Executes during object creation, after setting all
    properties.
function pos_valvula1_CreateFcn(hObject, eventdata, handles)
%hObject    handle to pos_valvula1 (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    empty - handles not created until after all
    CreateFcns called
```

```
%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

end
%--- Executes on slider movement.
function pos_valvula2_Callback(hObject, eventdata, handles)
posicion_valvula2=get(handles.pos_valvula2,'value');
handles.a.analogWrite(10,round((posicion_valvula2*255)/100));


%hObject    handle to pos_valvula2 (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%       get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider

end
%--- Executes during object creation, after setting all
    properties.
function pos_valvula2_CreateFcn(hObject, eventdata, handles)
%hObject    handle to pos_valvula2 (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    empty - handles not created until after all
    CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
end

%--- Executes on slider movement.
function pos_valvula3_Callback(hObject, eventdata, handles)
posicion_valvula3=get(handles.pos_valvula3,'value');
handles.a.analogWrite(8,round((posicion_valvula3*255)/100));


%hObject    handle to pos_valvula1 (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%       get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider

end
%--- Executes during object creation, after setting all
   properties.
function pos_valvula3_CreateFcn(hObject, eventdata, handles)
%hObject    handle to pos_valvula1 (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

end
%--- Executes on button press in estado_valvula_t_n.
function estado_valvula_t_n_Callback(hObject, eventdata,
   handles)
```

```
%hObject    handle to estado_valvula_t_n (see GCBO)
    activacion_todo_nada=get(hObject,'value');
if activacion_todo_nada==1
    handles.a.digitalWrite(24,1);
    set(handles.mos_valvulatn,'Visible','on');
else
     handles.a.digitalWrite(24,0);
        set(handles.mos_valvulatn,'Visible','off');
end

%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hint: get(hObject,'Value') returns toggle state of
   estado_valvula_t_n
end


%--- Executes on button press in menu.
function menu_Callback(hObject, eventdata, handles)
handles.a.analogWrite(2,0);
handles.a.analogWrite(4,0);
clear all; close all;clc; Pantalla_inicio;
%hObject    handle to menu (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)
end
```

10.5. Programa pantalla funcionamiento independiente

```
function varargout = depositos_independientes(varargin)
%DEPOSITOS_INDEPENDIENTES M-file for depositos_independientes.
   fig
%      DEPOSITOS_INDEPENDIENTES, by itself, creates a new
   DEPOSITOS_INDEPENDIENTES or raises the existing
```

```
%      singleton*.
%
%      H = DEPOSITOS_INDEPENDIENTES returns the handle to a new
    DEPOSITOS_INDEPENDIENTES or the handle to
%      the existing singleton*.
%
%      DEPOSITOS_INDEPENDIENTES('Property','Value',...) creates
    a new DEPOSITOS_INDEPENDIENTES using the
%      given property value pairs. Unrecognized properties are
   passed via
%      varargin to depositos_independientes_OpeningFcn. This
   calling syntax produces a
%      warning when there is an existing singleton*.
%
%      DEPOSITOS_INDEPENDIENTES('CALLBACK') and
   DEPOSITOS_INDEPENDIENTES('CALLBACK',hObject,...) call the
%      local function named CALLBACK in
   DEPOSITOS_INDEPENDIENTES.M with the given input
%      arguments.
%
%      *See GUI Options on GUIDE's Tools menu. Choose "GUI
   allows only one
%      instance to run (singleton)".
%
%See also: GUIDE, GUIDATA, GUIHANDLES

%Edit the above text to modify the response to help
   depositos_independientes

%Last Modified by GUIDE v2.5 21-Aug-2014 14:05:31

%Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
                      @depositos_independientes_OpeningFcn, ...
```

```
                    'gui_OutputFcn',
                      @depositos_independientes_OutputFcn, ...
                    'gui_LayoutFcn',  [], ...
                    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin
        {:});
else
    gui_mainfcn(gui_State, varargin{:});
end
%End initialization code - DO NOT EDIT

end
%--- Executes just before depositos_independientes is made
    visible.
function depositos_independientes_OpeningFcn(hObject, eventdata
    , handles, varargin)

%This function has no output args, see OutputFcn.
%hObject    handle to figure
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)
%varargin   unrecognized PropertyName/PropertyValue pairs from
    the
%            command line (see VARARGIN)
a=arduino('COM3');
a.pinMode(2,'output'); %bomba1
a.pinMode(4,'output');%bomba2
a.pinMode(8,'output');%valvula1
a.pinMode(10,'output');%valvula2
a.pinMode(12,'output');%valvula3
a.pinMode(24,'output');%valvula todo/nada
```

```
handles.a=a;
handles.a.analogWrite(2,0);
handles.a.analogWrite(4,0);
set(handles.mos_valvulatn,'Visible','off');
set(handles.parametros_pid,'Visible','on');
handles.Error_dcha=[0 0 0 ];
handles.Control_signal_dcha=[0 0];
handles.Error_izq=[0 0 0 ];
handles.Control_signal_izq=[0 0];
global output_value_izq %Inicialización del valor de salida
output_value_izq=0
global output_value_dcha %Inicialización del valor de salida
output_value_dcha=0
global Ti
global Td
global Kp
 Ti=7;
 Td=0.1;
 Kp=4;
 set(handles.valor_td,'value', Td);
set(handles.valor_ti,'value', Ti);
set(handles.valor_kp,'value', Kp);
set(handles.mos_kp,'string',Kp);
set(handles.mos_td,'string',Td);
set(handles.mos_ti,'string',Ti);
handles.n=1;
handles.m=1;
handles.orden_pid=1;
global valor_consigna_dcha
global valor_consigna_izq
global activacion_dcha
global activacion_izq
 valor_consigna_dcha=0;
 valor_consigna_izq=0;
 activacion_dcha=0;
 activacion_izq=0;
%Choose default command line output for
```

```
    depositos_independientes
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes depositos_independientes wait for user response
    (see UIRESUME)
% uiwait(handles.figure1);
tic;
end

%--- Outputs from this function are returned to the command
    line.
function varargout = depositos_independientes_OutputFcn(hObject
    , eventdata, handles)
global activacion_dcha
global activacion_izq
global Ti
global Td
global Kp
global valor_consigna_izq
global valor_consigna_dcha
global output_value_izq
global output_value_dcha
% varargout  cell array for returning output args (see
    VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
ant1_dcha=zeros(1,10000);
ant1_izq=zeros(1,10000);
ant2=0;
```

```
ant3=0;
antt=0;
ant2_izq=0;
ant3_izq=0;
antt_izq=0;


while(1)
    pause(0.005);
nivel_deposito_dcha=round(100-(((handles.a.analogRead(0)-
   handles.a.analogRead(2))*100)/1023));
nivel_deposito_izq=round(100-(((handles.a.analogRead(4)-handles
   .a.analogRead(6))*100)/1023));
set(handles.mos_dep_dcha,'string',fix(nivel_deposito_dcha));
set(handles.mos_dep_izq,'string',fix(nivel_deposito_izq));

input_value_dcha=handles.a.analogRead(0)-handles.a.analogRead
   (2);
input_value_izq=handles.a.analogRead(4)-handles.a.analogRead(6)
   ;

 if activacion_dcha==1 && handles.orden_pid==1
     handles.n=handles.n+1;
    handles.a.analogWrite(4,output_value_dcha);
    handles.input_voltage=round((nivel_deposito_dcha*1023)/100)
       ;
    [output_value_dcha,handles.Error_dcha,handles.
       Control_signal_dcha] = PID_PROPIO( Kp, Ti, Td,
       input_value_dcha,handles.Error_dcha,handles.
       Control_signal_dcha, valor_consigna_dcha)
    n_1=[(handles.n-1),handles.n];
    nivel_dep_1=[ant1_dcha(handles.n-1),ant1_dcha(handles.n),
       nivel_deposito_dcha];
    nivel_dep_2=[antt,nivel_deposito_dcha];
    potencia=[ant2,output_value_dcha*100/255];
     valor_consigna_1=[ant3,valor_consigna_dcha];
    ant1_dcha(handles.n)=nivel_deposito_dcha;
```

```
    antt=nivel_deposito_dcha;
    ant2=(output_value_dcha*100)/255;
    ant3=valor_consigna_dcha;
    [g_est_dcha]=rls_2polos(valor_consigna_1,nivel_dep_1)
set(handles.identificacion_dcha,'string',evalc('g_est_dcha'));
  axes(handles.grafica_dcha);
     plot(n_1,nivel_dep_2,'g-');
      hold on;
    plot(n_1, valor_consigna_1,'b-');
  plot(n_1,(potencia),'r-');
end

if activacion_izq==1 && handles.orden_pid==1
    handles.m=handles.m+1;
    handles.a.analogWrite(2,output_value_izq);
    handles.input_voltage=round((nivel_deposito_izq*1023)/100);
    [output_value_izq,handles.Error_izq,handles.
       Control_signal_izq] = PID_PROPIO( Kp, Ti, Td,
       input_value_izq,handles.Error_izq,handles.
       Control_signal_izq, valor_consigna_izq)
    n_1_1=[(handles.m-1),handles.m];
    nivel_dep_1_1=[ant1_izq(handles.m-1),ant1_izq(handles.m),
       nivel_deposito_izq];
    nivel_dep_2_1=[antt_izq,nivel_deposito_izq];
    potencia_1=[ant2_izq,output_value_izq*100/255];
     valor_consigna_1_1=[ant3_izq,valor_consigna_izq];
    ant1_izq(handles.n)=nivel_deposito_izq;
    antt_izq=nivel_deposito_izq;
    ant2_izq=(output_value_izq*100)/255;
    ant3_izq=valor_consigna_izq;
    [g_est_izq]=rls_2polos(valor_consigna_1_1,nivel_dep_1_1)
  set(handles.identificacion_izq,'string',evalc('g_est_izq'));
  axes(handles.grafica_izq);
     plot(n_1_1,nivel_dep_2_1,'g-');
      hold on;
    plot(n_1_1, valor_consigna_1_1,'b-');
  plot(n_1_1,(potencia_1),'r-');
```

```
 end

end

end

%--- Executes on button press in menu.
function menu_Callback(hObject, eventdata, handles)
handles.a.analogWrite(2,0);
handles.a.analogWrite(4,0);
clear all; close all;clc; Pantalla_inicio;
%hObject    handle to menu (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)
end

%--- Executes on slider movement.
function valor_ti_Callback(hObject, eventdata, handles)
global Ti

 Ti=get(handles.valor_ti,'value');
 set(handles.mos_ti,'string',Ti);
%hObject    handle to valor_ti (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%       get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider
end

%--- Executes during object creation, after setting all
   properties.
function valor_ti_CreateFcn(hObject, eventdata, handles)
%hObject    handle to valor_ti (see GCBO)
```

```
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
end

%--- Executes on slider movement.
function valor_td_Callback(hObject, eventdata, handles)
global Td

 Td=get(handles.valor_td,'value');
  set(handles.mos_td,'string',Td);
%hObject    handle to valor_td (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%       get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider
end

%--- Executes during object creation, after setting all
   properties.
function valor_td_CreateFcn(hObject, eventdata, handles)
%hObject    handle to valor_td (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called
```

```
%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
end

%--- Executes on slider movement.
function valor_kp_Callback(hObject, eventdata, handles)
global Kp

 Kp=get(handles.valor_kp,'value');
  set(handles.mos_kp,'string',Kp);
%hObject    handle to valor_kp (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%       get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider
end

%--- Executes during object creation, after setting all
    properties.
function valor_kp_CreateFcn(hObject, eventdata, handles)
%hObject    handle to valor_kp (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    empty - handles not created until after all
    CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
end


function mos_ti_Callback(hObject, eventdata, handles)
% hObject    handle to mos_ti (see GCBO)
% eventdata  reserved - to be defined in a future version of
   MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of mos_ti as
   text
%        str2double(get(hObject,'String')) returns contents of
   mos_ti as a double
end

%--- Executes during object creation, after setting all
   properties.
function mos_ti_CreateFcn(hObject, eventdata, handles)
% hObject    handle to mos_ti (see GCBO)
% eventdata  reserved - to be defined in a future version of
   MATLAB
% handles    empty - handles not created until after all
   CreateFcns called

% Hint: edit controls usually have a white background on
   Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function mos_td_Callback(hObject, eventdata, handles)
% hObject    handle to mos_td (see GCBO)
% eventdata  reserved - to be defined in a future version of
```

```
    MATLAB
%handles      structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'String') returns contents of mos_td as
    text
%        str2double(get(hObject,'String')) returns contents of
    mos_td as a double
end

%--- Executes during object creation, after setting all
    properties.
function mos_td_CreateFcn(hObject, eventdata, handles)
%hObject    handle to mos_td (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    empty - handles not created until after all
    CreateFcns called

%Hint: edit controls usually have a white background on
    Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function mos_kp_Callback(hObject, eventdata, handles)
%hObject    handle to mos_kp (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'String') returns contents of mos_kp as
    text
%        str2double(get(hObject,'String')) returns contents of
```

```
    mos_kp as a double
end

%--- Executes during object creation, after setting all
    properties.
function mos_kp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to mos_kp (see GCBO)
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    empty - handles not created until after all
    CreateFcns called

% Hint: edit controls usually have a white background on
    Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end
end

%--- Executes on button press in restablecer_pid.
function restablecer_pid_Callback(hObject, eventdata, handles)
global Ti
global Td
global Kp

 Ti=7;
 set(handles.valor_ti,'value', Ti);
  Td=0.01;
set(handles.valor_td,'value', Td);
  Kp=4;
set(handles.valor_kp,'value', Kp);

 set(handles.mos_kp,'string',Kp);
     set(handles.mos_td,'string',Td);
   set(handles.mos_ti,'string',Ti);
```

```
%hObject    handle to restablecer_pid (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)
end

%--- Executes on selection change in modo_control.
function modo_control_Callback(hObject, eventdata, handles)
global activacion_dcha
global activacion_izq
global Ti
global Td
global Kp

v=get(handles.modo_control,'value');

switch v
    case 1
        set(handles.parametros_pid,'Visible','on');

  Ti=7;
 set(handles.valor_ti,'value', Ti);
  Td=0.1;
set(handles.valor_td,'value', Td);
  Kp=4;

set(handles.valor_kp,'value', Kp);
  set(handles.mos_kp,'string',Kp);
    set(handles.mos_td,'string',Td);
  set(handles.mos_ti,'string',Ti);
    case 2
        handles.orden_pid=0;
        set(handles.parametros_pid,'Visible','on');

 if  activacion_dcha==1 | activacion_izq==1
     seleccion=0;
        [ Kp, Td, Ti ] = metodo_rele(handles.a,seleccion);
```

```
            handles.Orden_pid=1;
            set(handles.valor_ti,'value', Ti);
            set(handles.valor_td,'value', Td);
            set(handles.valor_kp,'value', Kp);
            set(handles.mos_kp,'string',Kp);
        set(handles.mos_td,'string',Td);
      set(handles.mos_ti,'string',Ti);
    end
        otherwise

end
%hObject    handle to modo_control (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: contents = cellstr(get(hObject,'String')) returns
   modo_control contents as cell array
%        contents{get(hObject,'Value')} returns selected item
   from modo_control
end

%--- Executes during object creation, after setting all
   properties.
function modo_control_CreateFcn(hObject, eventdata, handles)
%hObject    handle to modo_control (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: popupmenu controls usually have a white background on
   Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end
end

%--- Executes on slider movement.
function pos_valvula3_Callback(hObject, eventdata, handles)
posicion_valvula3=get(handles.pos_valvula3,'value');
handles.a.analogWrite(8,round((posicion_valvula3*255)/100));
%hObject    handle to pos_valvula3 (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%       get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider
end

%--- Executes during object creation, after setting all
   properties.
function pos_valvula3_CreateFcn(hObject, eventdata, handles)

%hObject    handle to pos_valvula3 (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
end

%--- Executes on slider movement.
function pos_valvula2_Callback(hObject, eventdata, handles)
posicion_valvula2=get(handles.pos_valvula2,'value');
```

```
handles.a.analogWrite(10,round((posicion_valvula2*255)/100));
%hObject    handle to pos_valvula2 (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider
end

%--- Executes during object creation, after setting all
   properties.
function pos_valvula2_CreateFcn(hObject, eventdata, handles)
%hObject    handle to pos_valvula2 (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
end

%--- Executes on slider movement.
function pos_valvula1_Callback(hObject, eventdata, handles)
posicion_valvula1=get(handles.pos_valvula1,'value');
handles.a.analogWrite(12,round((posicion_valvula1*255)/100));
%hObject    handle to pos_valvula1 (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
```

```
%        get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider
end

%--- Executes during object creation, after setting all
   properties.
function pos_valvula1_CreateFcn(hObject, eventdata, handles)

%hObject    handle to pos_valvula1 (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
end

%--- Executes on button press in estado_valvula_t_n.
function estado_valvula_t_n_Callback(hObject, eventdata,
   handles)
    activacion_todo_nada=get(hObject,'value');
if activacion_todo_nada==1
    handles.a.digitalWrite(24,1);
    set(handles.mos_valvulatn,'Visible','on');
else
     handles.a.digitalWrite(24,0);
        set(handles.mos_valvulatn,'Visible','off');
end
%hObject    handle to estado_valvula_t_n (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)
```

```
%Hint: get(hObject,'Value') returns toggle state of
    estado_valvula_t_n
end

%--- Executes on button press in paro_dcha.
function paro_dcha_Callback(hObject, eventdata, handles)
global activacion_dcha

    activacion_paro_dcha= get(hObject,'Value');
if  activacion_paro_dcha==1
      activacion_dcha=0;
      axes(handles.grafica_dcha);
      hold off;
      handles.a.analogWrite(4,0);


end
%hObject    handle to paro_dcha (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

end
%--- Executes on button press in arranque_dcha.
function arranque_dcha_Callback(hObject, eventdata, handles)
global activacion_dcha

 activacion_arranque_dcha= get(hObject,'Value');
if  activacion_arranque_dcha==1
      activacion_dcha=1
else
      activacion_dcha=0

      handles.a.analogWrite(4,0);
        axes(handles.grafica_dcha);
      hold off;
end
```

```
% hObject    handle to arranque_dcha (see GCBO)
% eventdata  reserved - to be defined in a future version of
   MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
   arranque_dcha

end
%--- Executes on button press in paro_izq.
function paro_izq_Callback(hObject, eventdata, handles)
global activacion_izq

   activacion_paro_izq= get(hObject,'Value');
if  activacion_paro_izq==1
     activacion_izq=0;
     handles.a.analogWrite(2,0);
      axes(handles.grafica_izq);
     hold off;


end
% hObject    handle to paro_izq (see GCBO)
% eventdata  reserved - to be defined in a future version of
   MATLAB
% handles    structure with handles and user data (see GUIDATA)
end

%--- Executes on button press in arranque_izq.
function arranque_izq_Callback(hObject, eventdata, handles)
global activacion_izq

 activacion_arranque_izq= get(hObject,'Value');
if  activacion_arranque_izq==1
     activacion_izq=1
else
     activacion_izq=0
```

```
        handles.a.analogWrite(2,0);
         axes(handles.grafica_izq);

        hold off;
end
%hObject    handle to arranque_izq (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hint: get(hObject,'Value') returns toggle state of
   arranque_izq
end

function mos_dep_izq_Callback(hObject, eventdata, handles)
%hObject    handle to mos_dep_izq (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'String') returns contents of mos_dep_izq
   as text
%        str2double(get(hObject,'String')) returns contents of
   mos_dep_izq as a double
end

%--- Executes during object creation, after setting all
   properties.
function mos_dep_izq_CreateFcn(hObject, eventdata, handles)
%hObject    handle to mos_dep_izq (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: edit controls usually have a white background on
   Windows.
```

```
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function mos_consigna_izq_Callback(hObject, eventdata, handles)
%hObject    handle to mos_consigna_izq (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'String') returns contents of
    mos_consigna_izq as text
%       str2double(get(hObject,'String')) returns contents of
    mos_consigna_izq as a double
end

%--- Executes during object creation, after setting all
    properties.
function mos_consigna_izq_CreateFcn(hObject, eventdata, handles
    )
%hObject    handle to mos_consigna_izq (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    empty - handles not created until after all
    CreateFcns called

%Hint: edit controls usually have a white background on
    Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
end

%--- Executes on slider movement.
function consigna_izq_Callback(hObject, eventdata, handles)
global valor_consigna_izq
consigna_izq=get(handles.consigna_izq,'value');
valor_consigna_izq=consigna_izq
set(handles.mos_consigna_izq,'string',fix(valor_consigna_izq));
%hObject    handle to consigna_izq (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%       get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider

end
%--- Executes during object creation, after setting all
   properties.
function consigna_izq_CreateFcn(hObject, eventdata, handles)

%hObject    handle to consigna_izq (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
end


function mos_dep_dcha_Callback(hObject, eventdata, handles)
```

```
%hObject    handle to mos_dep_dcha (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'String') returns contents of mos_dep_dcha
    as text
%        str2double(get(hObject,'String')) returns contents of
   mos_dep_dcha as a double

end
%--- Executes during object creation, after setting all
   properties.
function mos_dep_dcha_CreateFcn(hObject, eventdata, handles)
%hObject    handle to mos_dep_dcha (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: edit controls usually have a white background on
   Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function mos_consgina_dcha_Callback(hObject, eventdata, handles
   )
%hObject    handle to mos_consgina_dcha (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)
```

```
%Hints: get(hObject,'String') returns contents of
    mos_consgina_dcha as text
%        str2double(get(hObject,'String')) returns contents of
    mos_consgina_dcha as a double
end

%--- Executes during object creation, after setting all
    properties.
function mos_consgina_dcha_CreateFcn(hObject, eventdata,
    handles)
%hObject    handle to mos_consgina_dcha (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    empty - handles not created until after all
    CreateFcns called

%Hint: edit controls usually have a white background on
    Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

%--- Executes on slider movement.
function consigna_dcha_Callback(hObject, eventdata, handles)
global valor_consigna_dcha
consigna_dcha=get(handles.consigna_dcha,'value');
valor_consigna_dcha=consigna_dcha
set(handles.mos_consgina_dcha,'string',fix(valor_consigna_dcha)
    );
%hObject    handle to consigna_dcha (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)
```

```
%Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider

end
%--- Executes during object creation, after setting all
   properties.
function consigna_dcha_CreateFcn(hObject, eventdata, handles)
%hObject    handle to consigna_dcha (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
end
```

10.6. Programa pantalla funcionamiento entrelazados

```
function varargout = deposito_bomba_entrelazados(varargin)
%DEPOSITO_BOMBA_ENTRELAZADOS MATLAB code for
   deposito_bomba_entrelazados.fig
%      DEPOSITO_BOMBA_ENTRELAZADOS, by itself, creates a new
   DEPOSITO_BOMBA_ENTRELAZADOS or raises the existing
%      singleton*.
%
%      H = DEPOSITO_BOMBA_ENTRELAZADOS returns the handle to a
   new DEPOSITO_BOMBA_ENTRELAZADOS or the handle to
%      the existing singleton*.
%
%      DEPOSITO_BOMBA_ENTRELAZADOS('CALLBACK',hObject,eventData
   ,handles,...) calls the local
%      function named CALLBACK in DEPOSITO_BOMBA_ENTRELAZADOS.M
```

```
    with the given input arguments.
%
%      DEPOSITO_BOMBA_ENTRELAZADOS('Property','Value',...)
   creates a new DEPOSITO_BOMBA_ENTRELAZADOS or raises the
%      existing singleton*.  Starting from the left, property
   value pairs are
%      applied to the GUI before
   deposito_bomba_entrelazados_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes
   property application
%      stop.  All inputs are passed to
   deposito_bomba_entrelazados_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI
   allows only one
%      instance to run (singleton)".
%
%See also: GUIDE, GUIDATA, GUIHANDLES

%Edit the above text to modify the response to help
   deposito_bomba_entrelazados

%Last Modified by GUIDE v2.5 18-Aug-2014 11:35:19

%Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
                      @deposito_bomba_entrelazados_OpeningFcn,
                      ...
                   'gui_OutputFcn',
                      @deposito_bomba_entrelazados_OutputFcn,
                      ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
```

```
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin
        {:});
else
    gui_mainfcn(gui_State, varargin{:});
end
%End initialization code - DO NOT EDIT

end
%--- Executes just before deposito_bomba_entrelazados is made
    visible.
function deposito_bomba_entrelazados_OpeningFcn(hObject,
    eventdata, handles, varargin)

% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to
    deposito_bomba_entrelazados (see VARARGIN)
a=arduino('COM3');
a.pinMode(2,'output'); %bomba1
a.pinMode(4,'output');%bomba2
a.pinMode(8,'output');%valvula1
a.pinMode(10,'output');%valvula2
a.pinMode(12,'output');%valvula3
a.pinMode(24,'output');%valvula todo/nada
handles.a=a;
handles.a.analogWrite(2,0);
handles.a.analogWrite(4,0);
set(handles.mos_valvulatn,'Visible','off');
set(handles.parametros_pid,'Visible','on');
handles.Error=[0 0 0 ];
```

```
handles.Control_signal=[0 0];
global output_value %Inicialización del valor de salida
output_value=0
global Ti
global Td
global Kp
 Ti=9;
 Td=0.1;
 Kp=4;
    set(handles.valor_td,'value', Td);
   set(handles.valor_ti,'value', Ti);
   set(handles.mos_kp,'string',Kp);
     set(handles.mos_td,'string',Td);
   set(handles.mos_ti,'string',Ti);
handles.n=1;
handles.orden_pid=1;
global valor_consigna
global activacion
 valor_consigna=0
 activacion=0;
%Choose default command line output for
   deposito_bomba_entrelazados
handles.output = hObject;

%Update handles structure
guidata(hObject, handles);

%UIWAIT makes deposito_bomba_entrelazados wait for user
   response (see UIRESUME)
%uiwait(handles.figure1);
tic;
end
%--- Outputs from this function are returned to the command
   line.
function varargout = deposito_bomba_entrelazados_OutputFcn(
   hObject, eventdata, handles)
global activacion
```

```
global Ti
global Td
global Kp
global valor_consigna
global output_value
%varargout  cell array for returning output args (see
   VARARGOUT);
%hObject    handle to figure
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)
varargout{1} = handles.output;
%Get default command line output from handles structure
ant1=zeros(1,10000);
ant2=0;
ant3=0;
antt=0;
while(1)
    pause(0.005);
nivel_deposito_dcha=round(100-(((handles.a.analogRead(0)-
   handles.a.analogRead(2))*100)/1023));
set(handles.mos_dep_dcha,'string',fix(nivel_deposito_dcha));

input_value=handles.a.analogRead(0)-handles.a.analogRead(2)
 if activacion==1 && handles.orden_pid==1
     handles.n=handles.n+1;
    handles.a.analogWrite(2,output_value);
    handles.input_voltage=round((nivel_deposito_dcha*1023)/100)
       ;
    [output_value,handles.Error,handles.Control_signal] =
       PID_PROPIO( Kp, Ti, Td,input_value,handles.Error,handles
       .Control_signal, valor_consigna)
    n_1=[(handles.n-1),handles.n];
    nivel_dep_1=[ant1(handles.n-1),ant1(handles.n),
       nivel_deposito_dcha];
    nivel_dep_2=[antt,nivel_deposito_dcha];
    potencia=[ant2,output_value*100/255];
```

```
      valor_consigna_1=[ant3,valor_consigna];
     ant1(handles.n)=nivel_deposito_dcha;
     antt=nivel_deposito_dcha;
     ant2=(output_value*100)/255;
     ant3=valor_consigna;
     [g_est]=rls_2polos(valor_consigna_1,nivel_dep_1)
    set(handles.identificacion,'string',evalc('g_est'));
      plot(n_1,nivel_dep_2,'g-');
       hold on;
     plot(n_1, valor_consigna_1,'b-');
    plot(n_1,(potencia),'r-');
 end
end

end
%--- Executes on button press in menu.
function menu_Callback(hObject, eventdata, handles)
handles.a.analogWrite(2,0);
handles.a.analogWrite(4,0);
clear all; close all;clc; Pantalla_inicio;
% hObject    handle to menu (see GCBO)
% eventdata  reserved - to be defined in a future version of
   MATLAB
end
%--- Executes on slider movement.
function valor_ti_Callback(hObject, eventdata, handles)

global Ti

 Ti=get(handles.valor_ti,'value');
 set(handles.mos_ti,'string',Ti);
% hObject    handle to valor_ti (see GCBO)
% eventdata  reserved - to be defined in a future version of
   MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
```

```
%        get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider
end

%--- Executes during object creation, after setting all
   properties.
function valor_ti_CreateFcn(hObject, eventdata, handles)
%hObject    handle to valor_ti (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor',[.9 .9 .9]);
end

end
%--- Executes on slider movement.
function valor_td_Callback(hObject, eventdata, handles)

global Td

 Td=get(handles.valor_td,'value');
  set(handles.mos_td,'string',Td);

%hObject    handle to valor_td (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider
end
```

```
%--- Executes during object creation, after setting all
    properties.
function valor_td_CreateFcn(hObject, eventdata, handles)
% hObject    handle to valor_td (see GCBO)
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    empty - handles not created until after all
    CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor',[.9 .9 .9]);
end

end
%--- Executes on slider movement.
function valor_kp_Callback(hObject, eventdata, handles)

global Kp

 Kp=get(handles.valor_kp,'value');
  set(handles.mos_kp,'string',Kp);

% hObject    handle to valor_kp (see GCBO)
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine
     range of slider
end

%--- Executes during object creation, after setting all
    properties.
```

```
function valor_kp_CreateFcn(hObject, eventdata, handles)
%hObject    handle to valor_kp (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
end

%--- Executes on button press in restablecer_pid.
function restablecer_pid_Callback(hObject, eventdata, handles)

global Ti
global Td
global Kp

 Ti=9;
 set(handles.valor_ti,'value', Ti);
  Td=0.1;
set(handles.valor_td,'value', Td);
  Kp=4;
set(handles.valor_kp,'value', Kp);

 set(handles.mos_kp,'string',Kp);
    set(handles.mos_td,'string',Td);
  set(handles.mos_ti,'string',Ti);
%hObject    handle to restablecer_pid (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

end
```

```
%--- Executes on slider movement.
function pos_valvula3_Callback(hObject, eventdata, handles)
posicion_valvula3=get(handles.pos_valvula3,'value');
handles.a.analogWrite(8,round((posicion_valvula3*255)/100));
%hObject    handle to pos_valvula3 (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%       get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider
end

%--- Executes during object creation, after setting all
   properties.
function pos_valvula3_CreateFcn(hObject, eventdata, handles)
%hObject    handle to pos_valvula3 (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

end
%--- Executes on slider movement.
function pos_valvula2_Callback(hObject, eventdata, handles)
posicion_valvula2=get(handles.pos_valvula2,'value');
handles.a.analogWrite(10,round((posicion_valvula2*255)/100));
%hObject    handle to pos_valvula2 (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
```

```
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider
end

%--- Executes during object creation, after setting all
   properties.
function pos_valvula2_CreateFcn(hObject, eventdata, handles)
%hObject    handle to pos_valvula2 (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
end

%--- Executes on slider movement.
function pos_valvula1_Callback(hObject, eventdata, handles)
posicion_valvula1=get(handles.pos_valvula1,'value');
handles.a.analogWrite(12,round((posicion_valvula1*255)/100));
%hObject    handle to pos_valvula1 (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider

end
```

```
%--- Executes during object creation, after setting all
    properties.
function pos_valvula1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pos_valvula1 (see GCBO)
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    empty - handles not created until after all
    CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor',[.9 .9 .9]);
end
end

%--- Executes on button press in estado_valvula_t_n.
function estado_valvula_t_n_Callback(hObject, eventdata,
    handles)
     activacion_todo_nada=get(hObject,'value');
if activacion_todo_nada==1
     handles.a.digitalWrite(24,1);
     set(handles.mos_valvulatn,'Visible','on');
else
      handles.a.digitalWrite(24,0);
         set(handles.mos_valvulatn,'Visible','off');
end
% hObject    handle to estado_valvula_t_n (see GCBO)
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
    estado_valvula_t_n
end

%--- Executes on selection change in modo_control.
```

```
function modo_control_Callback(hObject, eventdata, handles)
global activacion
global Ti
global Td
global Kp

v=get(handles.modo_control,'value');

switch v
    case 1
        set(handles.parametros_pid,'Visible','on');

  Ti=9;
 set(handles.valor_ti,'value', Ti);
  Td=0.1;
set(handles.valor_td,'value', Td);
  Kp=4;

set(handles.valor_kp,'value', Kp);
  set(handles.mos_kp,'string',Kp);
    set(handles.mos_td,'string',Td);
  set(handles.mos_ti,'string',Ti);
    case 2
        handles.orden_pid=0;
        set(handles.parametros_pid,'Visible','off');

 if  activacion==1
     seleccion=1;
        [  Kp, Td, Ti ] = metodo_rele(handles.a,seleccion);
         handles.Orden_pid=1;
         set(handles.mos_kp,'string',Kp);
    set(handles.mos_td,'string',Td);
  set(handles.mos_ti,'string',Ti);
 end
    otherwise

end
```

```
%hObject    handle to modo_control (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: contents = cellstr(get(hObject,'String')) returns
   modo_control contents as cell array
%        contents{get(hObject,'Value')} returns selected item
   from modo_control
end
%--- Executes during object creation, after setting all
   properties.
function modo_control_CreateFcn(hObject, eventdata, handles)
%hObject    handle to modo_control (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: popupmenu controls usually have a white background on
   Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

%--- Executes on button press in paro.
function paro_Callback(hObject, eventdata, handles)
global activacion

   activacion_paro= get(hObject,'Value');
if  activacion_paro==1
     activacion=0;
     handles.a.analogWrite(2,0);
     handles.a.analogWrite(4,0);
```

```
    hold off;

end
%hObject    handle to paro (see GCBO)
%eventdata  reserved - to be defined in a future version of
  MATLAB
%handles    structure with handles and user data (see GUIDATA)

end

%--- Executes on button press in arranque.
function arranque_Callback(hObject, eventdata, handles)
global activacion

 activacion_arranque= get(hObject,'Value');
if  activacion_arranque==1
    activacion=1
else
    activacion=0
    handles.a.analogWrite(2,0);
    handles.a.analogWrite(4,0);
    hold off;
end


%hObject    handle to arranque (see GCBO)
%eventdata  reserved - to be defined in a future version of
  MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hint: get(hObject,'Value') returns toggle state of arranque
end


function mos_dep_dcha_Callback(hObject, eventdata, handles)
%hObject    handle to mos_dep_dcha (see GCBO)
%eventdata  reserved - to be defined in a future version of
```

```
    MATLAB
%handles      structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'String') returns contents of mos_dep_dcha
    as text
%        str2double(get(hObject,'String')) returns contents of
    mos_dep_dcha as a double
end
%--- Executes during object creation, after setting all
    properties.
function mos_dep_dcha_CreateFcn(hObject, eventdata, handles)
%hObject    handle to mos_dep_dcha (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    empty - handles not created until after all
    CreateFcns called

%Hint: edit controls usually have a white background on
    Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

end

function mos_consigna_izq_Callback(hObject, eventdata, handles)
%hObject    handle to mos_consigna_izq (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'String') returns contents of
    mos_consigna_izq as text
%        str2double(get(hObject,'String')) returns contents of
    mos_consigna_izq as a double
```

```
end
%--- Executes during object creation, after setting all
    properties.
function mos_consigna_izq_CreateFcn(hObject, eventdata, handles
    )
%hObject    handle to mos_consigna_izq (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    empty - handles not created until after all
    CreateFcns called

%Hint: edit controls usually have a white background on
    Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

%--- Executes on slider movement.
function consigna_izq_Callback(hObject, eventdata, handles)

global valor_consigna
consigna=get(handles.consigna_izq,'value');
valor_consigna=consigna
set(handles.mos_consigna_izq,'string',fix(consigna));
%hObject    handle to consigna_izq (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%       get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider
end
```

```
%--- Executes during object creation, after setting all
    properties.
function consigna_izq_CreateFcn(hObject, eventdata, handles)
% hObject    handle to consigna_izq (see GCBO)
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    empty - handles not created until after all
    CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor',[.9 .9 .9]);
end
end



function mos_ti_Callback(hObject, eventdata, handles)
% hObject    handle to mos_ti (see GCBO)
% eventdata  reserved - to be defined in a future version of
    MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of mos_ti as
    text
%        str2double(get(hObject,'String')) returns contents of
    mos_ti as a double

end
%--- Executes during object creation, after setting all
    properties.
function mos_ti_CreateFcn(hObject, eventdata, handles)
% hObject    handle to mos_ti (see GCBO)
% eventdata  reserved - to be defined in a future version of
    MATLAB
```

```
%handles    empty - handles not created until after all
    CreateFcns called

%Hint: edit controls usually have a white background on
    Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

end

function mos_td_Callback(hObject, eventdata, handles)
%hObject    handle to mos_td (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'String') returns contents of mos_td as
    text
%       str2double(get(hObject,'String')) returns contents of
    mos_td as a double

end
%--- Executes during object creation, after setting all
    properties.
function mos_td_CreateFcn(hObject, eventdata, handles)
%hObject    handle to mos_td (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    empty - handles not created until after all
    CreateFcns called

%Hint: edit controls usually have a white background on
    Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

end

function mos_kp_Callback(hObject, eventdata, handles)
%hObject    handle to mos_kp (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'String') returns contents of mos_kp as
    text
%        str2double(get(hObject,'String')) returns contents of
    mos_kp as a double
end

%--- Executes during object creation, after setting all
    properties.
function mos_kp_CreateFcn(hObject, eventdata, handles)
%hObject    handle to mos_kp (see GCBO)
%eventdata  reserved - to be defined in a future version of
    MATLAB
%handles    empty - handles not created until after all
    CreateFcns called

%Hint: edit controls usually have a white background on
    Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end
```

```
%--- Executes on slider movement.
function bomba_dcha_pot_Callback(hObject, eventdata, handles)
pot_bomba_dcha=get(handles.bomba_dcha_pot,'value');
handles.a.analogWrite(4,round((pot_bomba_dcha*255)/100));
set(handles.mos_bomba_dcha,'string',fix(pot_bomba_dcha));
%hObject    handle to bomba_dcha_pot (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'Value') returns position of slider
%       get(hObject,'Min') and get(hObject,'Max') to determine
    range of slider

end
%--- Executes during object creation, after setting all
   properties.
function bomba_dcha_pot_CreateFcn(hObject, eventdata, handles)
%hObject    handle to bomba_dcha_pot (see GCBO)
%eventdata  reserved - to be defined in a future version of
   MATLAB
%handles    empty - handles not created until after all
   CreateFcns called

%Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'
   defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
end
```

PLANOS

TÍTULO: **ESTRATEGIAS DE CONTROL PARA PLANTA DE LABORATORIO CON RETARDO**

PLANOS

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBRERO, S/N

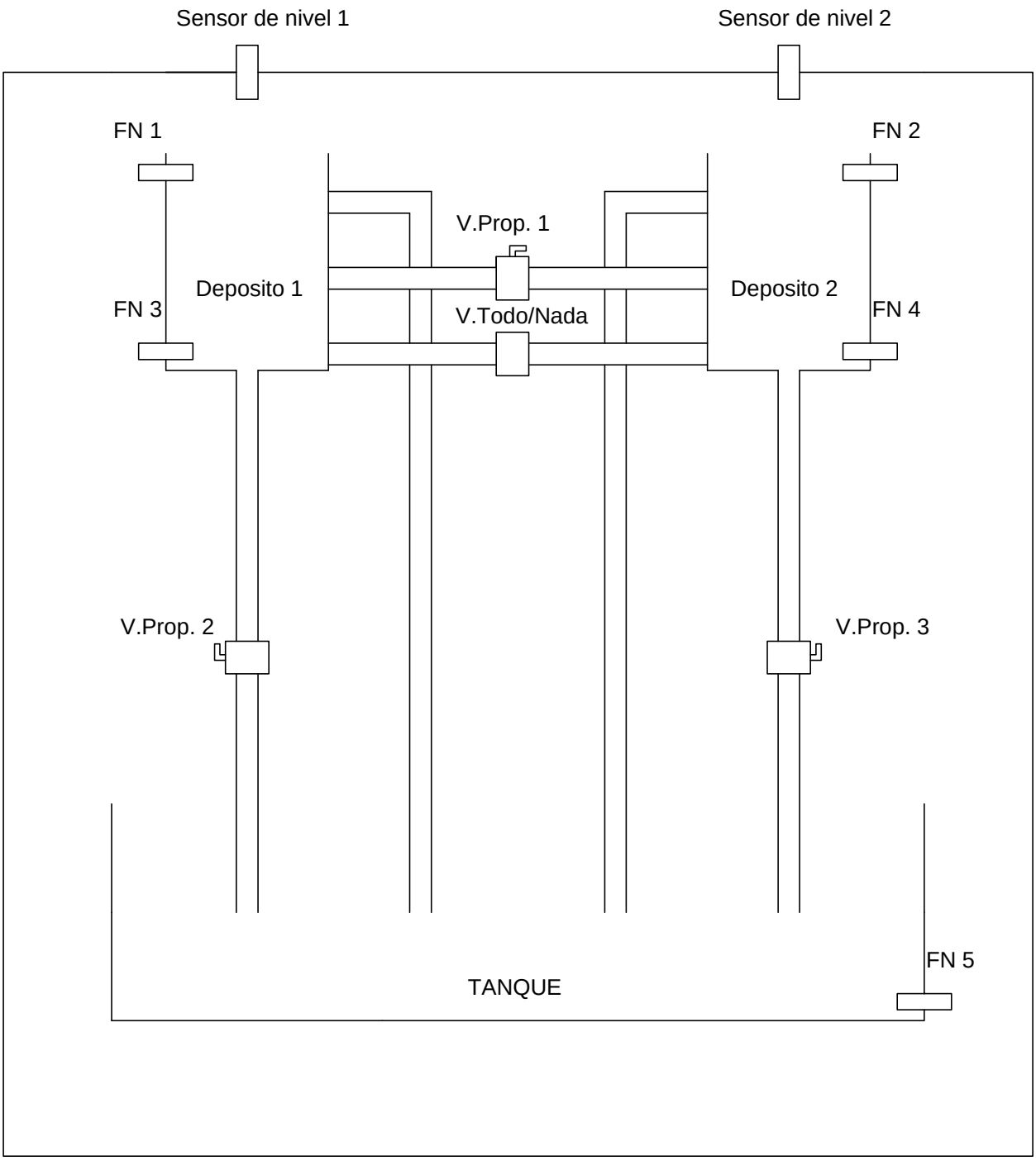
15405 - FERROL

DATA: SEPTIEMBRE DE 2014

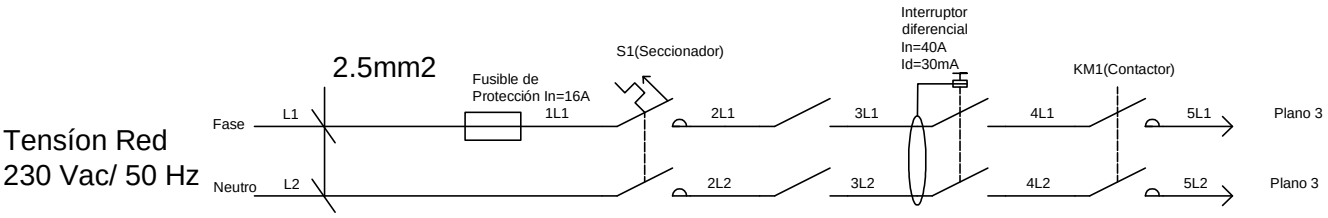
AUTOR: EL ALUMNO

Fdo.: RUBÉN VALIÑO DÍAZ

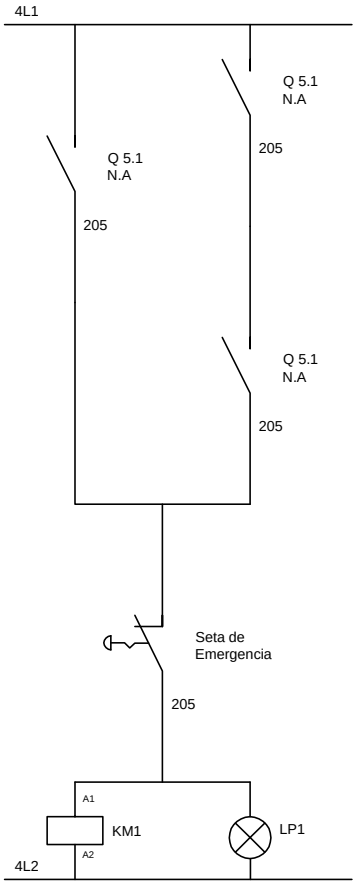
Índice de planos



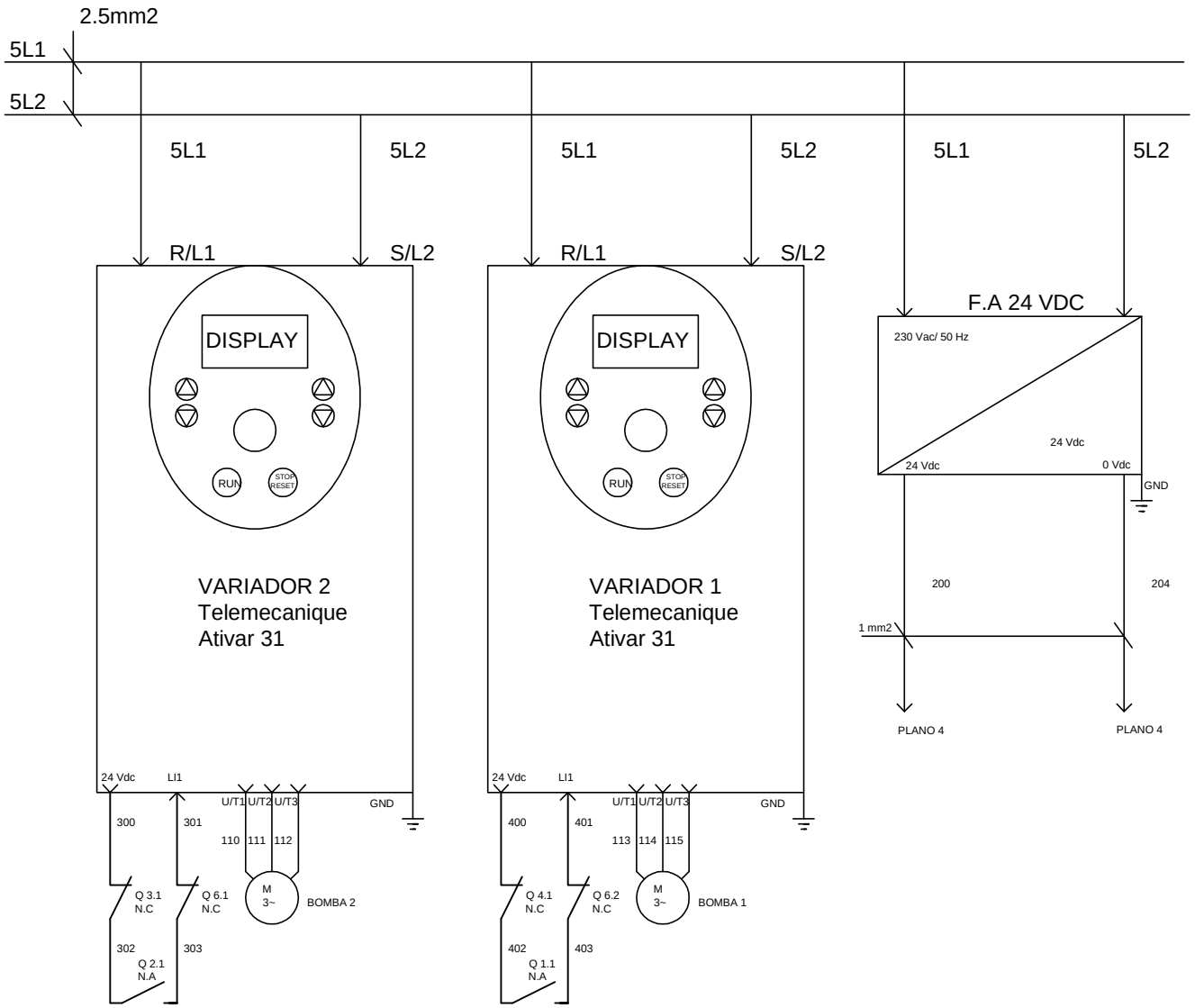
UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA	TFG Nº: 770G01A53
TÍTULO DEL TFG: Estrategias de control para Planta de Laboratorio con Retardo	
TÍTULO DEL PLANO: Descripción Planta de laboratorio	FECHA: SEPTIEMBRE-2014
	ESCALA:
AUTOR: Rubén valiño díaz	FIRMA:
	PLANO Nº: 01



Maniobra de mando



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA	TFG Nº: 770G01A53
TÍTULO DEL TFG: Estrategias de control para Planta de Laboratorio con Retardo	
TÍTULO DEL PLANO: Descripción Planta de laboratorio	FECHA: SEPTIEMBRE-2014 ESCALA:
AUTOR: Rubén valiño díaz FIRMA:	PLANO Nº: 02



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A53

TÍTULO DEL TFG:

Estrategias de control para Planta de Laboratorio con Retardo

TÍTULO DEL PLANO:

Plano general de alimentación

FECHA: SEPTIEMBRE-2014

ESCALA:

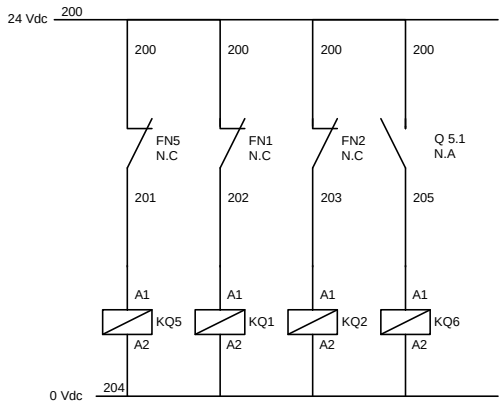
AUTOR:

Rubén valiño díaz

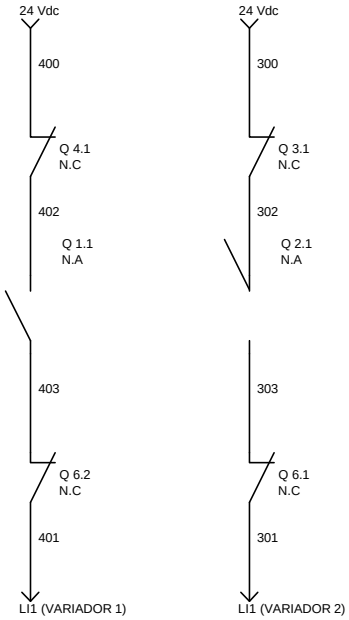
FIRMA:

PLANO Nº: 03

CONFIGURACIÓN FINALES DE CARRERA (FLOTADORES)



CONTACTOS PARA VARIADORES



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A53

TÍTULO DEL TFG:

Estrategias de control para Planta de Laboratorio con Retardo

TÍTULO DEL PLANO:

Configuración lógica de control

FECHA: SEPTIEMBRE-2014

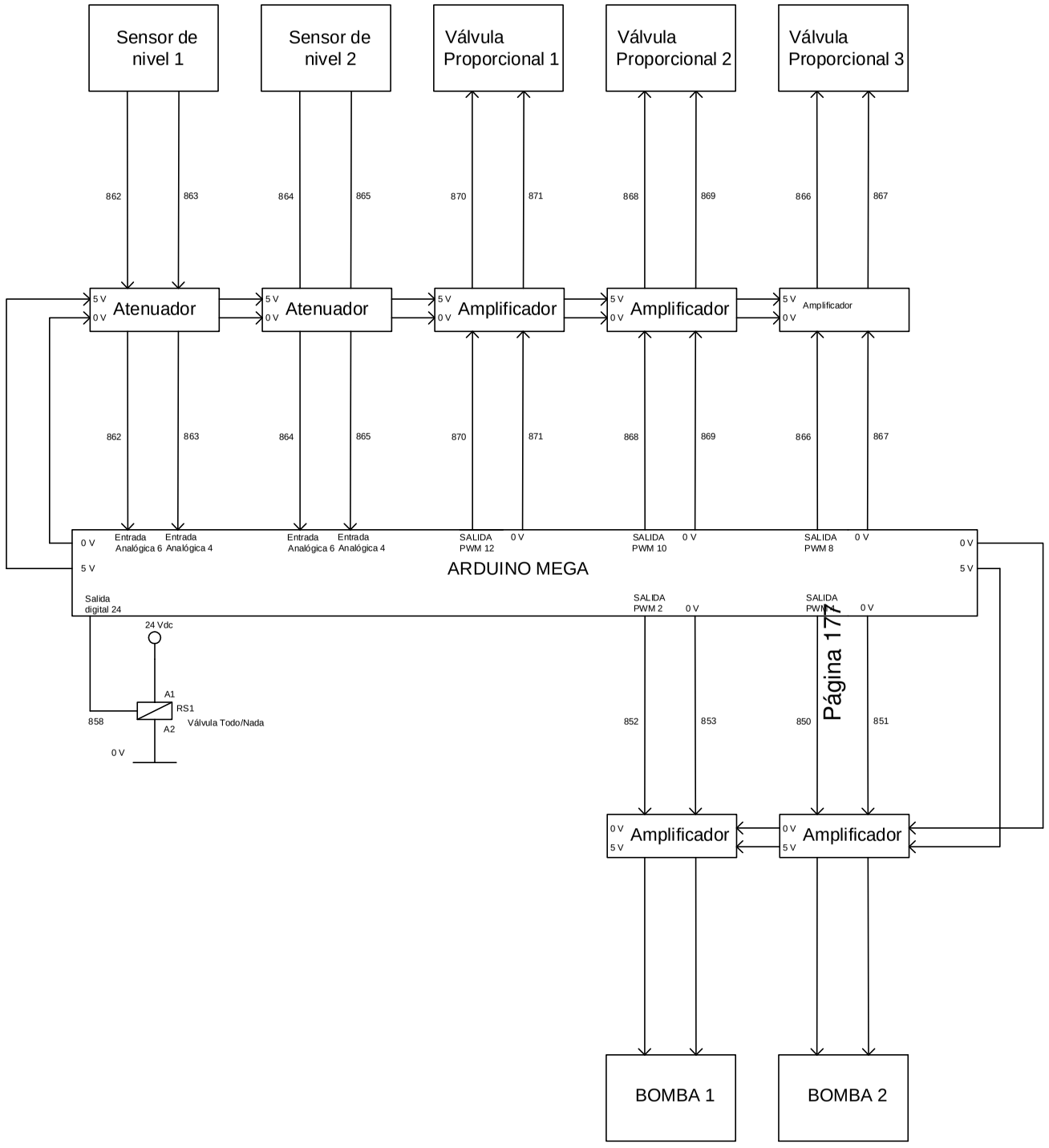
ESCALA:

AUTOR:

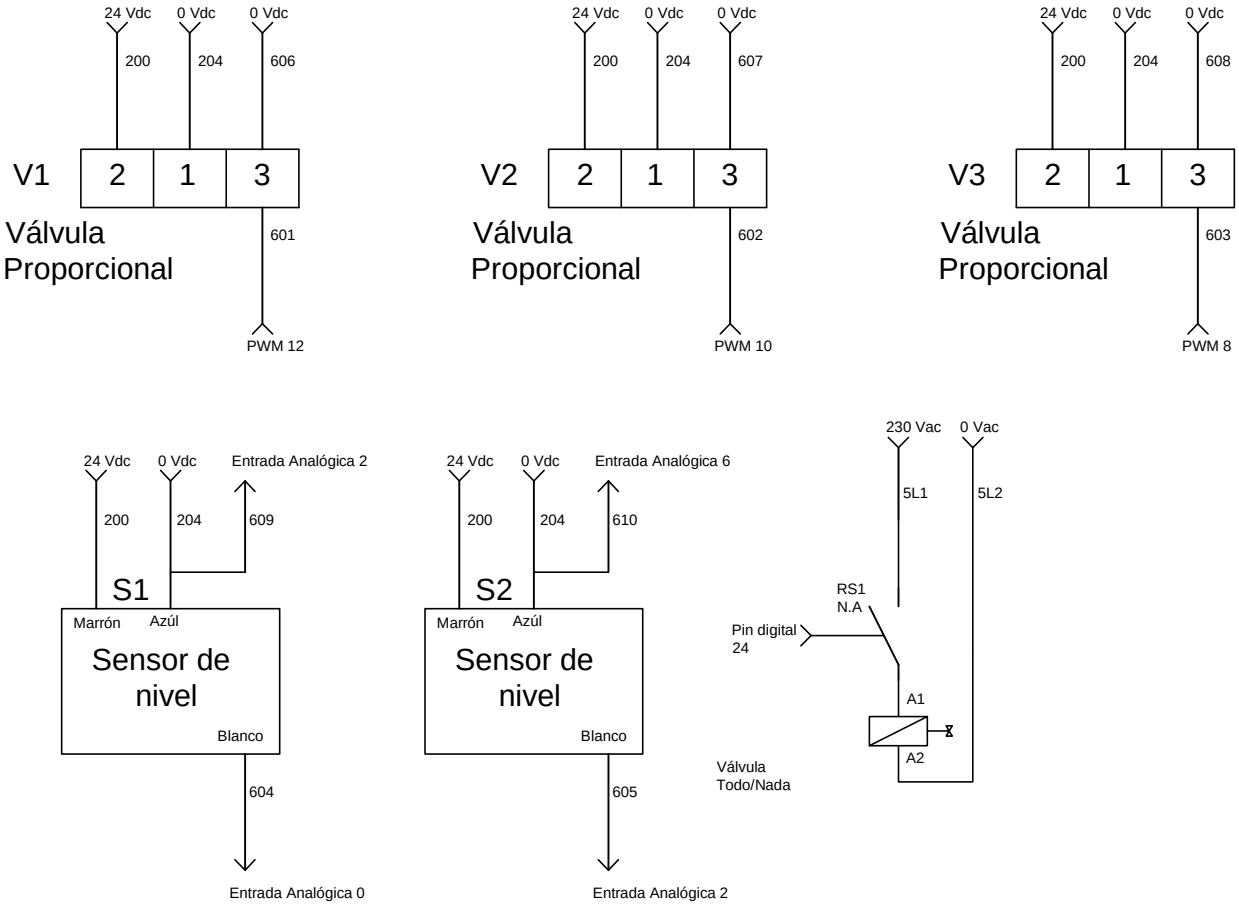
Rubén valiño díaz

FIRMA:

PLANO Nº: 04



ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA	TFG Nº: 770G01A53
TÍTULO DEL TFG: Estrategias de control para Planta de Laboratorio con Retardo	
TÍTULO DEL PLANO: Conexionado de amplificadores/atenuadores y Arduino	FECHA: SEPTIEMBRE-2014
	ESCALA:
AUTOR: Rubén valiño díaz	FIRMA:
	PLANO Nº: 05



UNIVERSIDADE DA CORUÑA

ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A53

TÍTULO DEL TFG:

Estrategias de control para Planta de Laboratorio con Retardo

TÍTULO DEL PLANO:

Plano conexionado válvulas y sensores

FECHA: SEPTIEMBRE-2014

ESCALA:

AUTOR:

Rubén valiño díaz

FIRMA:

PLANO Nº: 06

PLIEGO DE CONDICIONES

TÍTULO: **ESTRATEGIAS DE CONTROL PARA PLANTA DE LABORATORIO CON RETARDO**

PLIEGO DE CONDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBRERO, S/N

15405 - FERROL

DATA: SEPTIEMBRE DE 2014

AUTOR: EL ALUMNO

Fdo.: RUBÉN VALIÑO DÍAZ

Capítulo 11

Pliego de condiciones

11.1. Condiciones de uso

La planta de laboratorio debe estar situada en un lugar amplio en el que no exista ningún tipo de vapor para el correcto funcionamiento. Desde el instante en que se ponga en funcionamiento la puerta de la caja de conexiones debe estar cerrada para evitar los posibles riesgos eléctricos. Así mismo todo el conexionado de amplificadores y la conexión con la placa Arduino MEGA debe realizarse con la planta apagada y no modificarse durante el funcionamiento de la misma. El liquido a utilizar no debe tener una alta densidad ni viscosidad para el correcto funcionamiento de las bombas. Se recomiendo el uso de líquidos similares al agua, siendo el mas apropiado glicol para evitar la corrosión de los elementos metálicos. No se recomienda utilizar la planta en ámbientes peligrosos dado que el uso de Arduino no esta probado para estas atmósferas.

11.2. Selección de componentes de sustitución

En caso de exitir un deterioro de un elemento que no permita al mismo su función a de sustituirse con la planta apagada y desconectada de la red eléctrica. Los nuevos componentes deben poseer las mismas características de los actuales , en caso de no disponer de los mismos han de ser lo mas similares posibles.

11.3. Pruebas de comprobación

Antes de poner en funcionamiento la planta debe comprobarse que la red eléctrica a la que se conectará la planta cumple las condiciones mínimas. En un primer lugar debe tener una tensión de alimentación de 230 V y dar una intensidad mínima de 20

A. En caso de no soportar la instalación dicho valor de intensidad no debe procederse a la puesta en marcha de la planta.

Tras la conexión de la planta con el pc debe comprobarse que existe conexion entre ámbos. Para ello se debe en primer lugar comprobar a que puerta esta conectado el arduino así como si se dispone de los drivers necesarios. Posteriormente se comrpobará en la ventana de comandos de matlba que se ha producido la conexión y esta ha sido satisfactoria.

11.4. Requisitos de software y hardware

El sofware recomendado para el uso de la planta es la versión R2013A de matlab no asegurándose el correcto funcionamiento en versiones anteriores.

El equipo (pc) debe ser capaz de soportar las características de los software utilizados. El sistema operativo utilizado fue windows 8.

11.5. Almacenaje

En caso de almacenaje de la planta esta debe estar vacia, sin ningún líquido en los depósitos para evitar la posible corrosión de las partes metálicas. La temperatura de almacenaje no debe superar los 30^{o}C para evitar posibles degradaciones de los componentes.

ESTADO DE MEDICIONES

TÍTULO: **ESTRATEGIAS DE CONTROL PARA PLANTA DE LABORATORIO CON RETARDO**

ESTADO DE MEDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBRERO, S/N

15405 - FERROL

DATA: SEPTIEMBRE DE 2014

AUTOR: EL ALUMNO

Fdo.: RUBÉN VALIÑO DÍAZ

Capítulo 12

Estado de mediciones

12.1. Componentes planta

Elemento	Descripción	Unidades
Depósito	Elemento de almacenaje d liquidos 30L	2
Tanque	Elemento de almacenaje d liquidos 70L	1
Bomba	Bomba centrífuga Multinox de 4.8m3/h, presión 3,5 bares	2
Variador	Variador de velocidad Altivar 230 V - 0.75 kW	2
Válvula proporcional	Válvula Saeco,señal de control de 0-10 V configurable	3
Válvula todo/nada	Válvula normalmente cerrada Electrotaz 147L/h pilotable elécticamente	1
Flotador	Final de carrera 0-5 V de salida	5
Sensor de nivel	Sensor de ultrasonidos U-GAGE S18UUA, salida 0-10 v, resolución 1 mm	2

Tabla 12.1.0.1 – Componente planta

12.2. Aparamenta

Elemento	Descripción	Unidades
Seta de emergencia	Seta EMES 2.5 Kw	1
Interruptor	Interruptor marca Emes	2
Seccionador	Seccionador 2p 16 A	1
Contactor	Contactor marca MEC	1
Relé de estado sólido	Rele de 24 V	6
Temporizador	Temporizador MUR-3 configurable Crouzet	2
Interruptor diferencial	Interruptor diferencia 40 A, 30 mA	1
Interruptor magnetotérmico	Interruptor magnetotérmico 20 A	1
Interruptor general	Interruptor general 40 A	1
Fuente de alimentación	Fuente MEAN WELL 24 V	1
Relé 230V	Relé de 230 V	1

Tabla 12.2.0.2 – Aparamenta

12.3. Elementos externos

Elemento	Descripción	Unidades
Arduino Mega	Arduino MEGA rev 3	1
Cable USB	Cable USB tipo A-B	1
Amplificadores	Amplificadores de ganancia configurable 0.5 o 2	4

Tabla 12.3.0.3 – Elementos externos

PRESUPUESTO

TÍTULO: **ESTRATEGIAS DE CONTROL PARA PLANTA DE LABORATORIO CON RETARDO**

PRESUPUESTO

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBRERO, S/N

15405 - FERROL

DATA: SEPTIEMBRE DE 2014

AUTOR: EL ALUMNO

Fdo.: RUBÉN VALIÑO DÍAZ

Capítulo 13

Prespuesto

Para la realización de este proyecto se han utilizado los siguientes componentes y mano de obra. A la hora de realizar este presupuesto no se tiene en cuenta la planta de laboratorio dado que ya estaba constituida con anterioridad.

Concepto	Número de unidades	Precio Unitatio	Total
Arduino Mega	1	50 Euros/u	50
Cable USB	1	7 Euros/u	7
Horas de Ingeniería	240	40 Euros/h	9600
Horas de Técnico	20	25 Euros/h	500
Licencia Matlab R2013a	1	2000 Euros/u	500
TOTAL			**10657 Euros**

Tabla 13.0.0.1 – Presupuesto

El presupuesto para la realización del proyecto de la planta de laboratorio asciende a **10657 Euros** .